НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

“На правах рукопису”
УДК 519.688

«До захисту допущено»
Завідувач кафедри СПСКС
_____ В.П. Тарасенко
(підпис) (ініціали, прізвище)
“___” _____ 2018 р.

МАГІСТЕРСЬКА ДИСЕРТАЦІЯ

на здобуття ступеня магістра

зі спеціальності 123 «Комп'ютерна інженерія»
Системне програмування

на тему: «Тег-орієнтоване файлове сховище з прямим використанням накопичувача»

Виконав: студент II курсу, групи КВ-72мп
(шифр групи)
Мельник Олександр Олегович _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник к.т.н., доцент Потапова К.Р. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент: _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2018

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики
Кафедра системного програмування і спеціалізованих комп'ютерних систем)
Рівень вищої освіти — другий (магістерський)
Спеціальність 123 Комп'ютерна інженерія
Системне програмування

ЗАТВЕРДЖУЮ
Завідувач кафедри СПСКС
_____ В.П. Тарасенко
(підпис) (ініціали, прізвище)
"___" _____ 20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Мельнику Олександру Олеговичу
(прізвище, ім'я, по батькові)

1. Тема дисертації «Тег-орієнтоване файлове сховище з прямим використанням накопичувача» ,
науковий керівник дисертації Потапова Катерина Романівна, к.т.н, доцент ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від « 30 » жовтня 2018 р. №4030-с

2. Термін подання студентом дисертації 7 грудня 2018 р.

3. Об'єкт дослідження процеси організації асоціативно індексованого зберігання інформації.

4. Предмет дослідження побудова тег-орієнтованого файлового сховища на накопичувачі з використанням прямого доступу, а також управління вже побудованим таким сховищем.

5. Перелік завдань, які потрібно розробити
Створити метод побудови тег-орієнтованого файлового сховища з прямим використанням накопичувача
Розробити спосіб пакування часток файлу в незаповнений блок накопичувача
Розробити еталонну реалізацію методу

6. Перелік ілюстративного матеріалу презентація

_____

_____

7. Перелік публікацій _____

“Побудова тег-орієнтованого файлового сховища з прямим використанням накопичувача”, наукова конференція магістрантів та аспірантів “Прикладна математика та комп’ютинг” ПМК-2018 (Київ, 21-23 березня 2018 р.)

“Адаптивна модифікація методу побудови тег-орієнтованого файлового сховища”, наукова конференція магістрантів та аспірантів “Прикладна математика та комп’ютинг” ПМК-2018-2 (Київ, 14-16 листопада 2018 р.)

8. Дата видачі завдання 5 вересня 2017 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1	Вивчення літератури за тематикою проекту	01.12.2017	
2	Аналіз проблематики в за обраною тематикою	01.01.2018	
3	Аналіз існуючих рішень, підходів та методів	01.02.2018	
4	Підготовка матеріалів першого розділу магістерської дисертації	15.02.2018	
5	Створення власного методу	01.04.2018	
6	Дослідження та покращення власного методу	01.06.2018	
7	Підготовка матеріалів другого розділу магістерської дисертації	01.07.2018	
8	Підготовка матеріалів третього розділу магістерської дисертації	01.08.2018	
9	Підготовка матеріалів четвертого розділу магістерської дисертації	01.09.2018	
10	Підготовка ілюстративного матеріалу	01.10.2018	
11	Оформлення документації магістерської дисертації	01.11.2018	
12	Попередній розгляд магістерської дисертації на кафедрі	26.11.2018	

Студент _____ Мельник О.О.
(підпис) (ініціали, прізвище)

Керівник проекту _____ Потапова К.Р.
(підпис) (ініціали, прізвище)

РЕФЕРАТ

Актуальність теми. Незважаючи на наявність деякої кількості робіт, що розглядають тег-орієнтовані файлові сховища та системи, досі не було запропоновано методу побудови такого сховища з використанням прямого доступу до накопичувача, без проміжної ієрархічної файлової системи. Використання прямого доступу до накопичувача дозволяє виключити залежності від певного типу проміжної системи та виконати додаткову оптимізацію роботи файлового сховища. Метод побудови сховища з прямим доступом до накопичувача також дозволяє створити файлову систему на його основі. Запропонований метод є внеском у вирішення науково-технічної проблеми, актуальної для широкого спектру областей інформатики, адже файлове сховище чи система є невід’ємною компонентою будь-якої комп’ютерної системи.

Об’єктом дослідження є процеси організації асоціативно індексованого зберігання інформації.

Предметом дослідження є побудова тег-орієнтованого файлового сховища на накопичувачі з використанням прямого доступу, а також управління вже побудованим таким сховищем.

Мета роботи: створення методу, що дає змогу виконати побудову тег-орієнтованого файлового сховища на накопичувачі з використанням прямого доступу, без необхідності використання проміжної ієрархічної файлової системи, а також управління таким сховищем в подальшому.

Наукова новизна полягає в наступному:

1. Створено метод побудови тег-орієнтованого файлового сховища на накопичувачі з використанням прямого доступу.

2. Розроблено спосіб пакування часток файлу в незаповнений блок накопичувача.

Практична цінність отриманих в роботі результатів полягає в тому, що запропонований метод побудови тег-орієнтованого файлового сховища на накопичувачі з використанням прямого доступу, без необхідності використання проміжної ієрархічної файлової системи, яке може бути використане для збереження та управління наборами файлів. Теоретичні та практичні результати роботи можуть бути використані в організаціях, що потребують збереження, обробки та пошуку у великих об’ємах складно структурованих файлів.

Апробація роботи. Основні положення і результати роботи були представлені та обговорювались на науковій конференції магістрантів та аспірантів “Прикладна математика та комп’ютинг” ПМК-2018 (Київ, 21-23 березня 2018 р.) та на науковій конференції магістрантів та аспірантів “Прикладна математика та комп’ютинг” ПМК-2018-2 (Київ, 14-16 листопада 2018 р.)

Структура та обсяг роботи. Магістерська дисертація складається з вступу, чотирьох розділів та висновків.

У вступі подано загальну характеристику роботи, зроблено оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку досліджень, сформульовано мету і задачі досліджень, показано наукову новизну отриманих результатів і практичну цінність роботи, наведено відомості про апробацію результатів.

У першому розділі розглянуто проблематику та сценарії використання файлових сховищ та систем, існуючі методи їх вирішення, а також проведений аналіз, який дає змогу визначити основні переваги та недоліки цих методів. Розглянуті характеристики, за якими можна виконати

порівняння таких методів. Виявлено не вирішені, або недостатньо повно вирішені, проблеми.

У другому розділі наведено опис розробленого методу побудови тег-орієнтованого файлового сховища, формату накопичувача, таблиць індексації.

У третьому розділі описані структура і алгоритми програмної реалізації методу у вигляді модуля (бібліотеки) а також еталонної реалізації програмної системи, що використовує його. Наведені рекомендації щодо використання програмної системи та інтеграції модуля в інші програмні системи.

У четвертому розділі проводиться аналіз ефективності запропонованого методу та його еталонної реалізації в рамках виявлених у першому розділі проблем.

У висновках представлені результати проведеної роботи, дані рекомендації щодо використання методу.

Робота представлена на 90 аркушах, містить посилання на список використаних літературних джерел.

Ключові слова: тег, файлове сховище, накопичувач.

ABSTRACT

Relevance of the topic. Notwithstanding the existence of a few works on tag-based file storage, as of today no method of building such storage on a mass storage device, employing direct access to said device, without intermediary hierarchic file system, was proposed. Employment of the direct access to the mass storage device allows to eliminate dependencies of certain types of intermediary file system and to perform tag-based storage-specific optimizations. Method for the building of the tag-based storage directly on the mass storage device also allows the creation of a file system based on said storage in the course of further research. Proposed method contributes to the solution of the scientific and technical problem relevant for the wide plethora of fields in informatics, as file storage is an essential component of every computer system.

The object of research are the processes of organization of the associatively indexed information storage.

The subject of research is the building of tag-based file storage directly on mass storage device, employing direct access to said device, and also management of such storage previously built.

The purpose of the work: creation of a method that would allow to build tag-based file storage directly on mass storage device, employing direct access to said device, without intermediary hierarchic file system, and also further management of such storage.

The scientific novelty is as follows:

1. A method for building of tag-based file storage directly on mass storage device, employing direct access to said device, was created.

2. The method for packing parts of a file into a partially filled block of the mass storage device was created.

The practical value of the results obtained in the work is that a method of building of tag-based file storage directly on mass storage device, employing direct access to said device, without intermediary hierarchic file system, that can be used for storage and management of file sets, is proposed. Theoretical and practical results of the work may be used by the organizations that require storage, processing and searching in vast volume of complexly structured files.

Approbation of the work. The main provisions and results of the work were presented and discussed at the scientific conference of masters and postgraduates “Applied Mathematics and Computing”, PMK-2018 (Kyiv, March 21-23, 2018) and at the scientific conference of masters and postgraduates “Applied Mathematics and Computing”, PMK-2018-2 (Kyiv, November 14-16, 2018).

Structure and scope of the work. The master’s dissertation consists of an introduction, four sections and conclusions.

The introduction gives a general description of the work, assesses the current state of the problem, substantiates the relevance of the research direction, formulates the purpose and objectives of the research, shows the scientific novelty of the results obtained and the practical value of the work, provides information on the approbation of the results.

The first section examines the issues and scenarios of the file system and storage usage, existing methods of solution of the issues described, and also provides the analysis that allows to determine the main advantages and disadvantages of said methods. Yet unsolved, or not solved in complete entirety, problems are revealed.

The second section describes the proposed method for building of the tag-based storage, mass storage device format and the indexation tables.

The third section describes the structure and algorithms of the implementation of the proposed method in the form of a loadable software module (library) and the reference implementation of a software system employing the said module. Recommendations regarding the usage of the software system and the integration of the module into other software systems, are provided.

The fourth section performs the analysis of the proposed method's (and its reference implementation's) efficiency within the confines of the problems described in the first section.

In the conclusions the results of the work are presented, along with the recommendations regarding the usage of the method.

The work spans 90 sheets and contains references to the literary sources used.

Keywords: tag, file storage, mass storage device.

РЕФЕРАТ

Актуальность темы. Несмотря на наличие некоторого количества работ, рассматривающих тег-ориентированные файловые хранилища и системы, до сих пор не было предложено метода построения такого хранилища с использованием прямого доступа к накопителю, без промежуточной иерархической файловой системы. Использование прямого доступа к накопителю позволяет исключить зависимости от определённого типа промежуточной системы и выполнить дополнительную оптимизацию работы файлового хранилища. Метод построения хранилища с прямым доступом к накопителю также позволяет создать файловою систему на его основе. Предложенный метод является вкладом в решение научно-технической проблемы, актуальной для широкого спектра областей информатики, ведь файловое хранилище или система является неотъемлемой компонентой любой компьютерной системы.

Объектом исследования являются процессы организации ассоциативно индексированного хранения информации.

Предметом исследования является построение тег-ориентированного файлового хранилища на накопителе с использованием прямого доступа, а также управление уже построенным таким хранилищем.

Цель работы: создание метода, позволяющего выполнить построение тег-ориентированного файлового хранилища на накопителе с использованием прямого доступа, без необходимости использования промежуточной иерархической файловой системы, а также управление таким хранилищем в дальнейшем.

Научная новизна заключается в следующем:

1. Создан метод построения тег-ориентированного файлового хранилища на накопителе с использованием прямого доступа.

2. Разработано способ упаковывания частиц файла в незаполненный блок накопителя.

Практическая ценность полученных в работе результатов заключается в том, что предложен метод построения тег-ориентированного файлового хранилища на накопителе с использованием прямого доступа, без необходимости использования промежуточной иерархической файловой системы, которое может быть использовано для хранения и управления наборами файлов. Теоретические и практические результаты работы могут быть использованы в организациях, требующих хранения, обработки и поиска в больших объёмах сложно структурированных файлов.

Апробация работы. Основные положения и результаты работы были представлены и обсуждались на научной конференции магистрантов и аспирантов «Прикладная математика и компьютинг» ПМК-2018 (Киев, 21-23 марта 2018 г.) и на научной конференции магистрантов и аспирантов «Прикладная математика и компьютинг» ПМК-2018-2 (Киев, 14-16 ноября 2018 г.).

Структура и объём работы. Магистерская диссертация состоит из введения, четырёх глав и выводов.

В введении представлена общая характеристика работы, сделана оценка современного состояния проблемы, обоснована актуальность направления исследований, сформулирована цель и задачи исследований, показана научная новизна полученных результатов и практическая ценность работы, приведены сведения об апробации результатов.

В первой главе рассмотрены проблематика и сценарии использования файловых хранилищ и систем, существующие методы их решения, а также проведён анализ, позволяющий определить основные преимущества и

недостатки этих методов. Рассмотрены характеристики, по которым возможно выполнить сравнение таких методов. Определены не решённые, либо недостаточно полно решённые, проблемы.

Во второй главе приведено описание разработанного метода построения тег-ориентированного файлового хранилища, формата накопителя, таблиц индексации.

В третьей главе описаны структура и алгоритмы программной реализации метода в виде модуля (библиотеки), а также эталонной реализации программной системы, использующей его. Приведены рекомендации по использованию программной системы и интеграции модуля в иные программные системы.

В четвёртой главе проведён анализ эффективности предложенного метода и его эталонной реализации в рамках определённых в первой главе проблем.

В выводах представлены результаты проведённой работы, даны рекомендации по применению метода.

Работа представлена на 90 листах, содержит ссылки на список использованных литературных источников.

Ключевые слова: тег, файловое хранилище, накопитель.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ, ПОЗНАЧЕНЬ, ТЕРМІНІВ.....6

ВСТУП.....7

1. ПРОБЛЕМАТИКА І СЦЕНАРІЇ ВИКОРИСТАННЯ ФАЙЛОВИХ СХОВИЩ ТА СИСТЕМ.....8

1.1. Тег-орієнтований спосіб організації інформації в порівнянні з ієрархічним.....8

1.2. Існуючі реалізації тег-орієнтованих файлових сховищ та систем.10

1.3. Переваги використання прямого доступу до накопичувача.....11

1.4. Особливості використання накопичувача в режимі блокового пристрою.....13

1.5. Втрати простору накопичувача на кінцевих частинах файлів.....14

1.6. Фрагментація.....15

1.7. Зношування блоків flash-пам’яті.....17

1.8. Оптимізація за різними критеріями залежно від типу накопичувача.....18

1.9. Забезпечення цілісності інформації.....19

1.10. Захист від випадкового видалення.....20

1.11. Відновлення після раптової зупинки.....22

1.12. Використання декількох накопичувачів в рамках однієї файлової системи чи сховища.....23

2. МЕТОД ПОБУДОВИ ТЕГ-ОРІЄНТОВАНОГО ФАЙЛОВОГО СХОВИЩА НА НАКОПИЧУВАЧІ.....25

2.1. Основні положення.....25

2.2. Ділянка мета-даних.....26

2.3. Власне файлове сховище.....30

2.4. Кошик.....32

2.5. Розміщення таблиць у пам’яті.....33

2.6. Унікальна ідентифікація файлу.....33
2.7. Запобігання фрагментації.....34
2.8. Оптимізація за ресурсом запису фізичного блока.....37
2.9. Адаптивна природа оптимізації за ресурсом запису та запобігання фрагментації.....38
2.10. Пакування кінцевих частин файлу.....39
2.11. Журнал.....41
2.12. Забезпечення цілісності.....43
3. СТРУКТУРИ ДАНИХ І АЛГОРИТМИ ПРОГРАМНОЇ СИСТЕМИ.....45
3.1. Master-блок.....45
3.2. Заголовок ділянки мета-даних.....48
3.3. Заголовок журналу.....50
3.4. Таблиця адресації.....52
3.6. Таблиця файлових мета-даних.....55
3.7. Таблиця тегових мета-даних.....60
3.8. Перехресні таблиці асоціативної індексації.....62
3.9. Таблиця управління фрагментацією.....63
3.10. Таблиця обліку записів.....65
4. АНАЛІЗ РЕЗУЛЬТАТІВ.....66
4.1. Дослідження швидкодії вибірки файлів.....66
4.2. Дослідження використання простору накопичувача.....80
4.3. Дослідження використання ресурсу записів накопичувача.....84
ВИСНОВКИ.....88
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....90
ДОДАТКИ
Додаток 1. Копія довідки про впровадження
Додаток 2. Копія графічного матеріалу (презентація)

Додаток 3. Копія тез доповіді на тему: “Побудова тег-орієнтованого файлового сховища з прямим використанням накопичувача”

Додаток 4. Копія тез доповіді на тему: “Адаптивна модифікація методу побудови тег-орієнтованого файлового сховища”

Додаток 5. Фрагменти програмного тексту еталонної реалізації методу

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ, ПОЗНАЧЕНЬ, ТЕРМІНІВ

API — Application Programming Interface (інтерфейс програмування прикладних програм).

GID — Group Identifier (ідентифікатор групи).

GNU — Gnu is Not Unix (“GNU — не Unix”, власна назва і рекурсивна абревіатура).

FUSE — Filesystem in Userspace (файлова система на рівні користувача).

OS — Operating System (операційна система).

POSIX — Portable Operating System Interface (переносимий інтерфейс для операційних систем).

SHA3 — Secure Hash Algorithm 3 (алгоритм безпечного хешування 3).

UBIFS — Unsorted Block Image File System (файлова система для образів з несортованими блоками).

UID — User Identifier (ідентифікатор користувача).

Тег — властивість певного об’єкту, довільна мітка, що слугує мета-інформацією щодо даного об’єкту, корисною для користувача.

Прямий доступ до накопичувача — такий режим роботи з пристроєм накопичення інформації, за якого стає можливою поблочна адресація доступної пам’яті накопичувача, без використання абстракції файлової системи.

ВСТУП

Будь-яка комп’ютерна система (за винятком найпростіших вбудованих) оперує даними, зазвичай у вигляді або бази даних, або файлів. Відповідно, продуктивність будь-якої комп’ютерної системи може сильно змінитись в залежності від того, як саме виконана організація файлів — від структури файлової системи.

Найрозповсюдженіший сьогодні спосіб організації файлової системи — ієрархічний, зі вкладеними каталогами. Це простий у реалізації принцип, з яким можна досить зручно працювати програмно, але він має і деякі недоліки. Наприклад, така структура часто вимагає строго фіксованої організації файлів, а також не дає можливості їх впорядкування та вибірки за декількома критеріями одночасно.

За рахунок незначно більш складної та вимогливої щодо ресурсів системи реалізації можна нейтралізувати ці проблеми, використовуючи тег-орієнтоване файлове сховище. Але на сьогодні не існує жодної реалізації тег-орієнтованого файлового сховища на системному рівні (як повноцінної файлової системи), з прямим використанням накопичувача. Всі доступні реалізації використовують проміжну ієрархічну файлову систему, що вимагає додаткових витрат ресурсів та не дозволяє оптимізувати використання накопичувача.

В магістерській дисертації розглянуто саме метод побудови тег-орієнтованого файлового сховища з прямим використанням накопичувача, без проміжної файлової системи. Запропонований метод в подальшому можливо реалізувати на системному рівні ОС (операційної системи) як повноцінну файлову систему.

1. ПРОБЛЕМАТИКА І СЦЕНАРІЇ ВИКОРИСТАННЯ ФАЙЛОВИХ СХОВИЩ ТА СИСТЕМ

1.1. Тег-орієнтований спосіб організації інформації в порівнянні з ієрархічним

Історично комп’ютерні системи використовують ієрархічний спосіб організації інформації, тобто такий, що сформований у вигляді дерева, вузлами розгалуження якого є директорії, а кінцевими вузлами — файли. Ієрархічний спосіб побудови файлової системи є зручним та інтуїтивно зрозумілим у випадку, коли файли можуть бути однозначно організовані за рівнями вкладеності, проте не є оптимальним у випадку роботи з великими об’ємами одноманітних файлів (більшість розповсюджених сучасних файлових систем не здатні продуктивно працювати з директоріями, що містять десятки тисяч файлів або більше), а також у випадку коли така організація не є однозначною.

Нерідко файли мають декілька рівноправних ознак. Наприклад, для історичних документів це може бути дата формування, місце формування та автор. Можливо зберігати такі файли всі на одному рівні, розрізняючи ознаки за іменем файлу або його внутрішніми параметрами, але це робить пошук потрібних файлів за ними занадто вимогливим до ресурсів системи. Щоб зменшити навантаження на файлову систему, файли можливо зберігати за рівнями, обравши довільну їх структуру (наприклад, дата — перший рівень, місце — другий, автор — третій), але це, хоч пришвидшує вибірку, робить пошук за декількома критеріями (наприклад, документи, сформовані в певному місці певним автором), а також за критерієм одного з вкладених рівнів (наприклад, щоб отримати всі документи певного автора, необхідно виконати обхід всього дерева, перевіряючи кожне місце кожної дати) незручним. Також неможливо засобами файлової системи

виконати пошук за складним набором критеріїв (одне чи інше місце, або більш складний логічний вираз).

Сучасні файлові системи пропонують різноманітні способи для вирішення цих та інших подібних проблем, найвідоміший та найрозповсюдженіший серед яких — механізм “жорстких” та “м’яких” посилань на файли (в першому випадку в ієрархічному дереві файлової системи створюються цикли, і таким чином видається, начебто один і той самий файл знаходиться в декількох директоріях — є дочірньою вершиною для декількох батьківських; в другому випадку створюється спеціальний файл, що зберігає в собі шлях до іншого) — проте всі сценарії вирішення залишаються складними, громіздкими та/або незручними, адже є лише обхідними шляхами для виправлення недоліків самої структури ієрархічного дерева.

Інший такий недолік, що виправляється лише обхідними шляхами — необхідність визначити кореневий вузол дерева, “найвищу директорію”, що містить в собі всі інші. Така директорія не може фізично існувати (адже кожна директорія, як будь-який файл, повинна міститись в іншій), проте є необхідною для роботи з деревом. Операційні системи (ОС) сімейства Unix обирають спеціально визначений шлях “/” як корінь дерева основної файлової системи, а у випадку необхідності працювати з декількома файловими системами одночасно (наприклад, віддалений доступ до іншої комп’ютерної системи, або наявність в системі більш ніж одного накопичувача) одна з файлових систем вважається основною (і її корінь назначається директорією “/”), а корені інших під’єднуються (що в термінології Unix називають “монтуванням”) до вже існуючих в основній директорій. ОС сімейства Microsoft Windows, натомість, назначають кожній файловій системі літеру, таким чином додаючи ще один уявний рівень основного дерева, де розміщені (як літери) всі інші дерева файлових систем.

Всі ці, та багато інших, недоліки покликана виправити тег-орієнтована структура файлової системи. Найважливішим її аспектом є повна відмова від концепції “директорії” та вкладеності взагалі. Всі файли, розміщені в тег-орієнтованій (або, інакше, асоціативно індексованій) файловій системі зберігаються на одному рівні та не можуть бути вкладеними. Натомість, кожному з них може бути назначено довільну кількість будь-яких обраних користувачем міток (реалізовані як набір байтів, мітки можуть відображати як текстові рядки, так і, в загальному випадку, будь-яку іншу інформацію). Ці мітки називають “тегами” і використовують в подальшому для вибірки та пошуку бажаних файлів.

Якщо повернутись до прикладу з документами, переваги тег-орієнтованої файлової системи стають очевидними: кожен документ отримує три мітки (відповідно, місце формування, дата формування та автор), за якими потім його можна буде відшукати. Так як всі мітки знаходяться на одному рівні, пошук всіх документів певного автора буде виконуватись так само ефективно, як і пошук всіх документів певної дати, без необхідності обходу всього масиву файлів. Також стає можливим та ефективним в реалізації пошук за складними логічними виразами, що допускають наявність одного з декількох тегів, або вимагають наявності декількох одночасно, або виключають тег, або ж описують будь-яку комбінацію таких дій.

1.2. Існуючі реалізації тег-орієнтованих файлових сховищ та систем

На сьогодні існує кілька реалізацій тег-орієнтованих файлових сховищ, що заслуговують уваги: tag2find, TMSU, Tagsistant, Dantalian, DBFS, Tagfs, tagme-file.

Кожна з них має різні набори переваг та недоліків. Наприклад, всі наведені реалізації, окрім Dantalian, є самостійними програмами, тоді як

Dantalian є бібліотекою, тобто дозволяє інтеграцію з іншими програмами для спрощення доступу до сховища Dantalian. Всі вони, крім tag2find, є вільним програмним забезпеченням з відкритим програмним кодом та підтримують або всі популярні ОС, або якнайменше POSIX-сумісні ОС (tag2find підтримує лише Microsoft Windows). Рівень підтримки тег-орієнтованої функціональності різниться: так, tag2find, Tagsistant, DBFS та Tagfs не підтримують повноцінно складні запити (або лише прості, або обмежену кількість операцій), тоді як TMSU, Dantalian та tagme-file надають такі можливості. Серед існуючих реалізацій представлені різноманітні підходи до інтерфейсу: текстові (tagme-file, TMSU), графічні (tag2find, DBFS), FUSE (TMSU, Tagsistant) та API (Dantalian) інтерфейси пропонуються в залежності від обраного сховища. Особливої уваги заслуговує FUSE інтерфейс, що дозволяє працювати з тег-орієнтованим файловим сховищем так само, як і з ієрархічною файловою системою.

Проте основною та всезагальною для названих реалізацій рисою є використання ієрархічної файлової системи для побудови тег-орієнтованого сховища. Тобто, будь-яка тег-орієнтована операція відбувається на вищому рівні абстракції, ніж робота з накопичувачем, тоді як роботу з накопичувачем виконує ієрархічна файлова система. Різні реалізації використовують різні підходи щодо взаємодії з ієрархічною файловою системою (наприклад, Dantalian використовує жорсткі посилання, а tagme-file використовує структуру директорій з іменами, що відповідають префіксам хеш-сум збережених файлів), проте сам факт використання такої файлової системи в основі спостерігається в усіх випадках.

1.3. Переваги використання прямого доступу до накопичувача

Така організація тег-орієнтованого файлового сховища має як переваги, так і недоліки. Очевидними перевагами є відсутність

необхідності реалізації власних алгоритмів роботи з накопичувачем напряму (можливо використовувати стандартні методи ОС для роботи з файловою системою), підтримка всіх накопичувачів, що підтримуються файловими системами, що реалізують згадані стандартні методи ОС, а також прозорість для користувача (коли тег-орієнтована файлова структура побудована на базі ієрархічної файлової системи, користувач легко може переглянути її за допомогою будь-якого файлового менеджера).

Проте є й недоліки: використання певних особливостей файлової системи накладає вимоги щодо використання лише файлових систем (наприклад, якщо тег-орієнтоване сховище побудовано на жорстких посиланнях, як Dantalian, то його неможливо використовувати з ієрархічною файловою системою, що не підтримує такі посилання, в основі) з підтримкою таких особливостей, взагалі використання додаткової файлової системи вимагає додаткових затрат ресурсів (спочатку — на виконання тег-орієнтованого пошуку файлів, потів — на виконання ієрархічного їх пошуку, замість того щоб обмежитись тег-орієнтованим і виконати вибірку відразу напряму з накопичувача), не дозволяє побудувати оптимальну для тег-орієнтованого пошуку блокову структуру на накопичувачі, а також нерідко накладає обмеження на ОС, що можуть бути підтримувані таким тег-орієнтованим файловим сховищем (наприклад, ОС сімейства Unix та сімейства Microsoft Windows працюють, в загальному випадку, з різними ієрархічними файловими системами, тому підтримка обох цих сімейств є складнішою в реалізації для тег-орієнтованого файлового сховища).

Побудова тег-орієнтованого файлового сховища на накопичувачі з використанням прямого доступу дозволяє позбутися всіх описаних недоліків, тоді як описані переваги можна вважати менш значущими (а в багатьох випадках несуттєвими взагалі — наприклад, можливість переглянути ієрархічну внутрішню структуру сховища не є корисною для

пересічного користувача, а розповсюджених типів накопичувачів досить небагато, що дозволяє реалізувати підтримку лише їх, не зосереджуючи увагу на рідкісних, спеціалізованих типах). В даній магістерській дисертації запропоновано метод вирішення цієї задачі.

Також побудова тег-орієнтованого файлового сховища на накопичувачі з використанням прямого доступу дозволяє в подальшому реалізацію повноцінної файлової системи, побудованої на принципі асоціативної індексації (тег-орієнтованої). Файлова система може бути інтегрована в ОС для найбільш ефективного використання її можливостей. Проте реалізація файлової системи ставить додаткові складні задачі. Наприклад, підтримка стандартного інтерфейсу файлових систем, що побудований з врахуванням особливостей лише ієрархічних файлових систем і не надає всіх необхідних для зручної роботи з тег-орієнтованими файловими сховищами інструментів. Іншою складною задачею є створення методу трансляції ієрархічних файлових шляхів в тегові запити; при цьому бажано використовувати мінімум спеціальних символів, адже спеціальні символи часто є регламентованими на системному рівні та обмеженими у своїй кількості.

1.4. Особливості використання накопичувача в режимі блокового пристрою

В рамках використання прямого доступу до накопичувача необхідно вирішення низки проблем та врахування особливостей такого режиму роботи.

Першою важливою особливістю є використання накопичувача в режимі так званого блокового пристрою. Блоковий пристрій — розповсюджена (особливо в ОС сімейства Unix) система абстракції доступу до накопичувача, що представляє накопичувач у вигляді рядка байтів з можливістю прямої адресації. Найбільш значущою особливістю

цієї абстракції є те, що доступ до накопичувача виконується лише поблоково. Таким чином, адресувати можна лише початок блоку (якщо накопичувач використовує стандартний розмір блоку в 512 байт, це означає можливість використовувати адреси 0, 512, 1024 і так далі, тобто кратні 512, але ніякі інші). Також читати та записувати можна лише блок цілком.

Таким чином, при роботі з блоковим пристроєм необхідно враховувати розмір блоку, адресацію лише за блоками, та додавати/відтинати нульові байти при записі/читанні з накопичувача, необхідні для вирівнювання інформації під розмір блоку.

Адресація лише за блоками та необхідність вирівнювати інформацію, надану в об’ємах, не кратних розміру блоку, очевидно призводить до наступної проблеми, що вимагає вирішення: при записі файлу розміром навіть всього один байт, на накопичувачі буде зайнято повний блок. Якщо зберігати у сховищі велику кількість дрібних файлів, це може швидко призвести до зайвих помітних витрат пам’яті накопичувача. Деякі файлові системи (наприклад, ReiserFS) вирішують цю проблему через використання так званого хвостового пакування, коли при наявності двох файлів, що виходять розміром за кратне розміру блока число на сумарно менший чи рівний розміру блока об’єм, ця “зайва” інформація записується не в два блоки, а в один, “з голови” (тобто в початок блоку) та “з хвоста” (тобто в кінець блоку).

1.5. Втрати простору накопичувача на кінцевих частинах файлів

Для уникнення зайвих витрат дискового простору необхідно реалізувати метод хвостового пакування, або подібний. Проте необхідно враховувати, що це додає зайві витрати на обробку при читанні та записі файлів, тому режим хвостового пакування повинен вмикатися та вимикатися відповідно до побажань користувача.

1.6. Фрагментація

Широко відома проблема різноманітних файлових систем — фрагментація. В загальному випадку проблема фрагментації описується так: якщо на накопичувачі є два “острівці” послідовних вільних блоків, що сумарно відповідають за розміром файлу, файл може бути розрізано навпіл та записано у ці острівці, так що дві частини файлу будуть фізично розміщені в різних частинах накопичувача. Розміщення в різних частинах накопичувача вимагає, відповідно, декілька операцій фізичного пошуку засобами накопичувача, та в найгірших випадках (файл значного розміру розбито на сотні частин) призводить до дуже значного зниження продуктив1ності (швидкості читання/доступу тощо).

Різні файлові системи вирішують проблему фрагментації по-різному. Наприклад, файлові системи сімейства FAT добре відомі повною відсутністю будь-якого методу вирішення проблеми фрагментації. Користувачу пропонувалося періодично використовувати окрему утиліту дефрагментації, що перевіряла накопичувач та переміщувала блоки фрагментованих файлів в послідовні позиції. Схожа проблема існує у файлової системи HFS, що намагається відтворювати ієрархічну структуру фізично на накопичувачі, що неминуче призводить до фрагментації.

Файлові системи сімейства ext, натомість, намагаються завжди уникнути фрагментації, наскільки це можливо. Ці файлові системи розділяють файл на декілька частин лише якщо на накопичувачі абсолютно неможливо знайти такий острівець вільного місця, що вмістив би файл цілком. Проте навіть підхід ext має власні недоліки: у випадку дописування до вже існуючого файлу вільне місце на накопичувачі може закінчитись, призводячи до фрагментації таким чином. Інакше можна було б читати файл цілком та намагатися записати до нового місця, проте у випадку файлів значного розміру це викликало б неочевидні для користувача

затримки в роботі, тому при створенні файлових систем сімейства ext від цього підходу відмовились.

Інші файлові системи (наприклад, btrfs) використовують більш складні алгоритми уникання фрагментації (на основі двійкових дерев тощо), проте в рамках методу побудови тег-орієнтованого файлового сховища на накопичувачі з використанням прямого доступу цілком достатньо запровадити підхід, подібний до використаного в файлових системах сімейства ext (вибір, за можливості, такого острівця вільного місця, щоб файл не потрібно було фрагментувати).

Проблема фрагментації є актуальною не для всіх типів накопичувачів, а лише для тих, що потребують додаткових затрат на доступ до фізично розміщених не послідовно блоків. Наприклад, страждають від фрагментації дискові накопичувачі (жорсткі диски, оптичні диски, магнітні гнучкі диски) та стрічкові накопичувачі, проте не страждають електронні (flash-пам’ять, NAND та NOR пам’ять з прямим доступом, твердотілі накопичувачі тощо). Це досягається за рахунок того, що доступ до будь-якого блоку електронного накопичувача відбувається за однаковий, постійний проміжок часу, тоді як дисковому накопичувачу необхідно фізично виконати оборот робочого полотна та переміщення читаючої/пишучої голівки. Затрачений на оборот та переміщення голівки час залежить від швидкості двигуна та відстані, яку потрібно пройти, тому ніколи не може бути постійним (і однозначно не може бути таким же або меншим, як аналогічний час пошуку блоку для електронного накопичувача). Навіть при послідовному читанні чи записі без фрагментації, позбавлені залежності від фізичних процесів обертання та переміщення у просторі електронні накопичувачі показують значно вищу продуктивність, що не знижується навіть у випадку значної фрагментації файлу.

Проте твердотілі накопичувачі та flash-пам’ять мають і власні недоліки. Найзначніший з них — зношування окремих фізичних блоків.

1.7. Зношування блоків flash-пам’яті

Для дискових накопичувачів характерний вихід з ладу накопичувача цілком. Фізичний блок в такому випадку є просто фізичною ділянкою магнітного диску, може бути перезаписаний довільну кількість разів, а основними причинами виходу з ладу накопичувача стають несправності моторів, що обертають диск чи переміщують магнітну голівку, самої магнітної голівки чи контролера. Єдиним випадком, коли виходять з ладу окремі блоки накопичувача, може бути фізичне пошкодження частини диску (наприклад, мікроподряпина часточкою пилу тощо), і зазвичай в момент, коли такі пошкодження стають можливими, весь накопичувач потрібно вважати несправним, адже статистичний прогноз щодо його подальшого строку служби є несприятливим.

Натомість, для твердотілих накопичувачів та flash-пам’яті кожен фізичний блок є фактично окремим електронним модулем, і кожен з них може вийти з ладу, ніяк не впливаючи на роботу інших. Єдиною причиною виходу з ладу всього накопичувача в такому випадку стає несправність електронного контролера, що має надійність значно вищу, ніж мотори дискового накопичувача та складна за конструкцією магнітна голівка.

Кожний блок такого накопичувача гарантує фактично необмежену кількість читань збереженої інформації, але обмежену кількість записів (від тисяч для найдешевших у виробництві накопичувачів до сотень тисяч у найнадійніших). Таким чином, часті перезаписи одного й того самого файлу будуть швидко зношувати накопичувач, особливо якщо враховувати особливість блокової адресації, що вимагає перезапис блоку цілком навіть для файлу, меншого об’ємом за блок. Проблема стає ще більш очевидною, якщо звернути увагу на особливості внутрішньої адресації

розповсюджених сьогодні твердотілих накопичувачів що, для здешевлення конструкції, використовують фізичні блоки (інакше відомі як “сторінки”) розміром навіть не 512 чи 4096 байт (популярні для дискових накопичувачів розміри фізичних блоків), а від 16 до 512 кібібайт в залежності від виробника, класу накопичувача тощо.

Тому у випадку використання сучасної flash-пам’яті або твердотілих накопичувачів контролер накопичувача надає операційній системі лише абстрактний блоковий інтерфейс, що не відповідає внутрішній структурі накопичувача (відомій лише контролеру). Таким чином контролер забезпечує рівномірне зношення накопичувача (виконуючи запис кожного разу у найменш зношений фізичний блок). Іншими словами, на фізичному рівні кожен сучасний накопичувач, побудований на flash-пам’яті або твердотілій технології, є дуже сильно фрагментованим, проте завдяки описаним особливостям роботи це ніяк не впливає на їх продуктивність.

1.8. Оптимізація за різними критеріями залежно від типу накопичувача

Отже, будуючи будь-яку (тег-орієнтовану чи ієрархічну) файлову систему чи сховище на накопичувачі, необхідно запобігати фрагментації у випадку, якщо це дисковий (або сконструйований за інакшим принципом, але такий, що вимагає більших затрат часу на доступ до фізично рознесених блоків) накопичувач, але не потрібно зважати на фрагментацію у випадку роботи з flash-пам’яттю чи твердотілим накопичувачем, адже в такому випадку контролер все одно не дозволяє доступ на достатньому для виконання. Крім того, фрагментація ніяк не впливає на роботу такого накопичувача. Додаткові затрати на її запобігання лише знизять продуктивність роботи з файловою системою чи сховищем.

І навпаки, при побудові файлової системи чи сховища на дисковому накопичувачі немає сенсу слідкувати за кількістю записів у конкретний

блок чи за ефективністю хвостового пакування блоків, тоді як у випадку flash-пам’яті чи твердотілого накопичувача кількість перезаписів блоку є дуже важливою, а хвостове пакування набуває цілком нового значення, адже не лише економить використаний простір накопичувача, а й запобігає зайвим перезаписам блоків. Однак, в реальних умовах часто неможливо дізнатись розмір фізичного блоку накопичувача, як і отримати доступ до нього на такому низькому рівні. Всі описані функції розробниками сучасної flash-пам’яті та твердотілих накопичувачів покладено на контролер. Розробнику файлової системи чи сховища залишається лише довіритись їх рекомендаціям та хіба що використовувати додаткове кешування даних, щоб уникнути занадто частих записів на накопичувач.

Нарешті, у випадку використання NAND чи NOR пам’яті з прямим доступом наявні всі переваги та недоліки flash-пам’яті та твердотілих накопичувачів (вища швидкість доступу; нерелевантність фрагментації щодо продуктивності роботи; зношення окремих блоків, а не накопичувача в цілому; обмежений ресурс запису і необмежений — читання), проте контролери зазвичай є набагато простішими та покладають задачі балансування ресурсу накопичувача на операційну та файлову системи. Наприклад, файлова система UBIFS (Unsorted Block Image File System, Файлова Система для Образів з Несортованими Блоками) призначена саме для вирішення цієї задачі. Отже, якщо в тег-орієнтованому сховищі планується підтримка таких видів накопичувачів, буде необхідним врахування цієї особливості їх роботи.

1.9. Забезпечення цілісності інформації

Будь-які пристрої збереження та передачі інформації допускають, з різною ймовірністю, її пошкодження в процесі запису, читання чи передачі — так звані бітові помилки (порушення сигналу через завади чи інші причини, коли один чи декілька бітів повідомлення міняють своє

значення). Майже всі сучасні накопичувачі забезпечують автоматичне запобігання та корекцію подібних помилок на рівні контролера шляхом використання стійких до завад кодувань з можливістю корекції, перевірок вмісту файлу за контрольними сумами тощо. Проте в деяких випадках реалізацій на рівні контролера виявляється недостатньо (чи, що сьогодні рідкісно, але можливо, накопичувач не підтримує їх взагалі) — щоб запобігти помилок в такому випадку, деякі файлові системи запроваджують додаткові перевірки коректності записаної інформації (зазвичай значення хеш-функції від змісту файлу чи його контрольну суму).

1.10. Захист від випадкового видалення

Важливою проблемою не на технічному, а на користувацькому рівні є випадкове видалення важливої інформації. Різні файлові системи використовують різні підходи для вирішення цієї проблеми.

Найрозповсюдженішим способом є повна відмова від її вирішення і передача цієї задачі на рівень вище — до операційної системи. За таким принципом працюють файлові системи сімейств FAT, ext тощо. Інформацію нерідко можливо відновити з накопичувача (адже, для пришвидшення роботи, на файл просто знищується посилання, а місце його розміщення вважається файловою системою вільним для запису — записаний на диск зміст файлу не перезаписується нульовими значеннями) за допомогою додаткових інструментів, але успішне відновлення гарантується лише за умови, що в файлову систему з моменту видалення не виконувалось ні одного запису, яка виконується в реальних умовах досить рідко.

Іншим способом може стати збереження посилань на “видалені” файли у наданому файловою системою сховищі — так званому “кошику”. Проте такий підхід підвищує ймовірність фрагментації файлів (адже дисковий простір, зайнятий видаленими файлами, насправді не

звільняється, і можливо, що новий файл буде необхідно розділити на декілька фрагментів саме через це).

Можливо також фізично переміщувати видалений файл у відокремлену ділянку накопичувача, проте такий підхід вимагає додаткових затрат часу на процедури видалення та відновлення, у випадку файлів великого розміру — значних. Також, подібний спосіб збереження видаленого файлу може вимагати для процедури видалення наявність острівця послідовного вільного дискового простору, що буває незручним для користувача у випадку видалення файлів великого розміру, або привносити додатковий ризик пошкодження файлу, якщо процедуру видалення буде перервано без можливості безпечно завершити її (наприклад, шляхом вимикання живлення). Такі вимоги зумовлено тим, що при перенесенні часток файлу в нове місце з одночасним їх видаленням, процедуру видалення неможливо перервати і відмінити швидко, а при раптовому її перериванні на накопичувачі залишаються дві рознесені частини файлу, що не завжди можуть бути безпечно об’єднані знову. Щоб запобігти цьому (надати можливість швидкої зупинки та відміни видалення, а також гарантувати відновлення файлу навіть у випадку раптової зупинки), файл необхідно копіювати в нове місце цілком і видаляти, лише впевнившись в успіху операції копіювання — тут, очевидно, постає вимога наявності необхідного вільного простору в ділянці “кошика”.

Можливо, оптимальним вирішенням проблеми відміни видалення стане запит до користувача, чи бажає він виконувати видалення з можливістю відновлення. Такий запит можна виконувати на рівні всієї файлової системи, або кожного файлу. Також можливо надавати його лише у певних випадках, наприклад, при перевищенні файлом, що видаляється, заданого розміру, або коли видалити файл безпечно (через процедуру копіювання в ділянку “кошика”) неможливо через брак вільного простору

на накопичувачі. Можливо і автоматично надавати можливість відновлення лише для невеликих файлів. Проте більшість файлових систем залишають цей вибір операційній системі, не надаючи можливостей для відновлення видалених файлів взагалі, або надаючи лише рудиментарні.

1.11. Відновлення після раптової зупинки

Інша значуща проблема, що постає відразу з декількох аспектів побудови та використання файлових систем та сховищ, це відновлення роботи після раптової зупинки (наприклад, втрата накопичувачем живлення). Цю проблему майже ніколи не вирішують на рівні накопичувача, тому саме файлові (а у випадку відсутності підтримки такої функціональності у файлової системи, операційні) системи мають виконувати перевірку власного стану, щоб впевнитись в коректності всіх збережених файлів та можливості продовжувати роботу без ризику для даних користувача.

Деякі файлові системи (наприклад, сімейства FAT) ніяк взагалі не вирішують дану проблему. Серед інших популярний такий метод як журналювання: файловою системою на накопичувачі ведеться журнал всіх запланованих та виконаних операцій з зазначенням того, з якими файлами та ділянками накопичувача вони були чи мають бути виконані. При завантаженні файлова система перевіряє журнал, і якщо знаходить невідповідності між записами в ньому та фактичною ситуацією на накопичувачі (або просто запис про намір виконання дії без підтвердження її успіху), виносить вердикт щодо некоректності файлової системи накопичувача, повідомляє про це операційну систему та користувача, після чого виконує виправлення. Деякі виправлення (перерване копіювання чи видалення, наприклад) можуть бути виконані за журналом (копіювання чи видалення просто виконується знову), тоді як інші (наприклад, запис файлу

з оперативної пам’яті) не можуть бути виконані і залишки некоректних файлів чи записи в журналі просто видаляються.

Журналювання є загалом ефективним вирішенням проблеми раптового завершення роботи, проте при використанні з електронними накопичувачами (flash-пам’ять, твердотілі накопичувачі, NAND та NOR пам’ять) додають значні додаткові витрати ресурсу накопичувача, адже запис до журналу має виконуватись перед початком та наприкінці будь-якої операції з накопичувачем. Тримати журнал в оперативній пам’яті також неможливо, адже це повністю знищує будь-які переваги від його використання (адже при раптовій втраті живлення вміст оперативної пам’яті втрачається). Для цієї проблеми зараз не відомо оптимального вирішення; у випадку використання електронних накопичувачів або відмовляються від журналювання, намагаючись запобігти пов’язаним з раптовим завершенням роботи проблемам інакше (як то: неможливість завершити роботу інакше, як безпечним шляхом за безпечний час, на рівні ОС; використання джерел безперебійного живлення тощо), або приймають необхідність додаткових витрат ресурсу накопичувача.

1.12. Використання декількох накопичувачів в рамках однієї файлової системи чи сховища

Особливої уваги заслуговують випадки, коли одночасно використовуються декілька накопичувачів. Класично в такому випадку кожний накопичувач має власну файлову систему та для надання користувачу доступу до кожного з них використовуються засоби операційної системи (в ОС сімейства Unix — монтування, в ОС сімейства Microsoft Windows — літери накопичувачів тощо), проте цей підхід має недоліки. Наприклад, на двох доступних системі накопичувачах може бути сумарно достатньо місця для розміщення певного файлу, проте на жодну з них окремо його не вдасться записати. Або ж може бути просто незручно

використовувати дві окремі файлові системи — відповідно різні літери чи директорії — для збереження однотипних файлів, наприклад документів, на обох, лише через те, що одної недостатньо, щоб вмістити весь об’єм файлів. Це ускладнює пошук необхідного файлу (потрібно перевірити обидві файлові системи тощо).

Набагато зручніше в цьому випадку побудувати одну файлову систему на декількох накопичувачах. Сучасні технології дозволяють декілька шляхів для досягнення цього (більшість з них, проте, доступна лише для операційних систем сімейства Unix), наприклад, реалізовані в апаратурі конфігурації RAID, реалізовані програмно конфігурації RAID, система LVM. Окремі файлові системи (наприклад, ZFS та btrfs) дозволяють використання декількох накопичувачів власними методами, без абстракції RAID.

2. МЕТОД ПОБУДОВИ ТЕГ-ОРІЄНТОВАНОГО ФАЙЛОВОГО СХОВИЩА НА НАКОПИЧУВАЧІ

2.1. Основні положення

Для побудови структури тег-орієнтованого файлового сховища на накопичувачі використовується логічний розмір блоку 4 КіБ (4096 байт), як найрозповсюдженіший в індустрії на даний момент. Переважна більшість сучасних дискових накопичувачів виготовляються з використанням фізичного розміру кластера 4 КіБ, а переважна більшість твердотілих накопичувачів використовують навіть більші сторінки для низькорівневої адресації. Доля накопичувачів, що використовують фактично застарілий стандартний розмір кластера 512 байт є незначною; до того ж, використання більшого розміру блока для адресації не є проблемним поки розмір блоку залишається кратним розміру фізичного кластера. Так як число 4096 є кратним 512, погіршення продуктивності на накопичувачах, що мають фізичний розмір кластера 512 байт, не очікується.

Накопичувач використовується сховищем повністю. Підтримка роботи з логічними розділами накопичувача не надається; пропонується або використовувати накопичувач цілком під єдине тег-орієнтоване файлове сховище, або заздалегідь виконати розмітку на логічні розділи сторонньою утилітою (наприклад, parted), так щоб в рамках програми управління тег-орієнтованим файловим сховищем, що реалізує запропонований метод, логічний розділ розглядався як окремий накопичувач.

Відповідно, в подальшому під “накопичувач” можна розуміти як, власне, цілком накопичувач, так і один логічний розділ накопичувача, залежно від підходу користувача до організації роботи з накопичувачами. Для програмних реалізацій методу дозволяється додавати підтримку роботи з логічними розділами накопичувача власними засобами. Також цей напрям є перспективним для подальшого розвитку методу.

Загальну структуру тег-орієнтованого файлового сховища, побудованого на накопичувачі з використанням прямого доступу, можна подати у вигляді послідовності трьох логічних ділянок:

— Мета-дані;

— Власне файлове сховище;

— Кошик.

Приклад такої реалізованої на накопичувачі структури наведено на рис. 2.1.

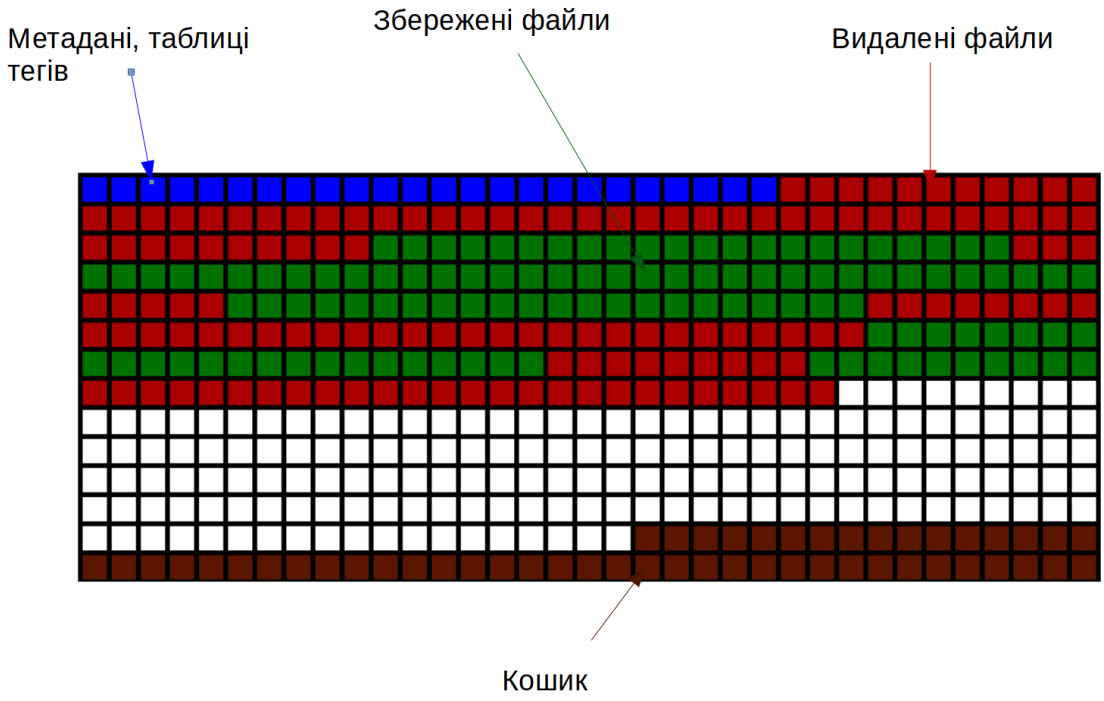

Рисунок 2.1 — Приклад структури тег-орієнтованого файлового сховища, побудованого на накопичувачі

2.2. Ділянка мета-даних

Ділянка мета-даних включає в себе наступні структури даних:

— Таблиця адресації;

— Таблиця файлових мета-даних;

— Таблиця тегових мета-даних;

— Перехресні таблиці асоціативної індексації (таблиця індексації тегами та таблиця індексації файлами);

— Таблиця індексації незайнятого простору накопичувача;

— Таблиця видалених файлів;

— Таблиця управління фрагментацією;

— Таблиця мета-даних для хвостового пакування;

— Таблиця обліку записів.

На розсуд реалізації віддається об’єднання таблиць. Наприклад, можуть бути об’єднані таблиці адресації та файлових мета-даних, таблиці мета-даних хвостового пакування і так далі. Найбільш виродженим випадком є об’єднання всіх таблиць лише у дві, що відповідають перехресним таблицям асоціативної індексації та містять в собі всі інші дані.

Таблиця адресації зберігає в собі записи, що становлять у відповідність внутрішній для сховища ідентифікатор файлу та його адресу на накопичувачі (або посилання до таблиці мета-даних для хвостового пакування чи таблиці управління фрагментацією, у випадку відповідно використання хвостового пакування для зберігання файлу та фрагментації файлу).

Таблиця файлових мета-даних зберігає в собі записи, що становлять у відповідність внутрішній для сховища ідентифікатор файлу та відповідні йому мета-дані. Наприклад, тут може зберігатись ім’я файлу, інформація щодо його типу, розширення чи походження, інформація щодо власника та прав доступу (зокрема, права в стилі Unix) тощо. Конкретний набір мета-даних визначається конкретною реалізацією методу; у випадку, коли реалізація не використовує мета-дані за винятком тегів, дану таблицю можна вилучити.

Таблиця тегових мета-даних зберігає в собі записи, що становлять у відповідність внутрішній для сховища ідентифікатор тегу та відповідні йому мета-дані. Наприклад, тут може зберігатись ілюстрація чи коментар до тегу. Обов’язковим типом мета-даних для тегів є так звані псевдоніми (англ. alias) — теги, що вважаються ідентичними до певного іншого тегу. Як приклад псевдоніму можна розглянути тег “пташка”, встановлений як псевдонім тегу “птах”.

Перехресні таблиці асоціативної індексації є ключовим механізмом для функціонування тег-орієнтованого сховища. Дві таблиці (таблиця індексації файлів тегами та таблиця індексації тегів файлами) надають можливість швидкого отримання як списку файлів, що мають певний тег, так і списку тегів, що присвоєні певному файлу, та в подальшому виконання операцій з цими списками відповідно до тегового запиту. Вміст таблиць відповідає їх призначенню: таблиця індексації файлів тегами використовує тег в якості індексу та список файлів в якості значення, а таблиця індексації тегів файлами — навпаки, файл в якості індексу та список тегів в якості значення.

Таблиця індексації незайнятого простору накопичувача є важливою складовою механізму управління простором накопичувача та дозволяє швидко визначити позицію для розміщення нового файлу на ньому завдяки тому, що зберігає індексовані посилання на вільні ділянки накопичувача з вказанням їх розміру. Для пришвидшення пошуку ділянки потрібного розміру та інших операцій ця таблиця має бути відсортованою за розміром ділянки вільного простору.

Таблиця видалених файлів дозволяє слідкувати за станом кожного файлу та реалізувати можливість відновлення останнього видаленого файлу. Ділянки накопичувача, зайняті видаленими файлами, розглядаються сховищем як такі, що мають деякий проміжний стан між зайнятими та вільними: за умови, що такий запис не спричинить значного погіршення

продуктивності сховища, запис нових файлів в ці ділянки не відбувається. Таблиця видалених файлів дозволяє, відповідно, вести облік таких ділянок та використовується в рамках алгоритму оцінки тої чи іншої позиції для нового файлу. В ній зберігається вся необхідна для такого використання (та відновлення видаленого файлу) інформація: розмір файлу, його розміщення, мета-дані, список тегів, тощо. Призначені видаленому файлу теги не будуть вважатись втраченими, допоки файл ще занесено до таблиці видалених файлів і не перезаписано. В момент перезапису файл вилучається з таблиці.

Таблиця управління фрагментацією дозволяє вести облік фрагментованих файлів (таких, що записані в дві або більше рознесених фізично ділянки накопичувача). Використання таблиці управління фрагментацією уможливлює, власне, використання фрагментації (що є нерідко єдиним достатньо швидким способом записати на накопичувач файл значного розміру, коли на накопичувачі загалом достатньо вільного простору для цього, але відсутні ділянки фізично послідовних вільних кластерів, тобто вільний простір є фізично рознесеним по накопичувачу у вигляді двох чи більше ділянок). Також таблиця управління фрагментацією дозволяє реалізацію динамічної дефрагментації (переміщення частин файлу в нову позицію на накопичувачі, як тільки з’являється ділянка достатнього розміру, в процесі його нормального використання) та спрощує реалізацію капітальної дефрагментації (виконується для всього накопичувача, не дозволяючи його використання в процесі). Інформація щодо фрагментації також використовується в алгоритмах оцінки позицій для файлів, вибору кандидатів з видалених файлів для перезапису тощо.

Таблиця мета-даних для хвостового пакування зберігає в собі дані, що дозволяють визначити, чи було використане хвостове пакування для певного файлу, і якщо так — в який кластер сумісно з яким іншим файлом запаковано його хвостову частину. За принципом організації та

використанням ця таблиця є схожою до таблиці управління фрагментацією, проте об’єднання з таблицею фрагментації не є однозначно доцільним, адже таблиці хвостового пакування та фрагментації оперують різними одиницями адресації (блоки та байти) та загалом оперують на різних рівнях.

Таблиця обліку записів зберігає в собі інформацію щодо того, скільки разів було перезаписано кожний підконтрольний сховищу блок накопичувача. Ця таблиця створюється лише у випадку побудови тег-орієнтованого файлового сховища на накопичувачі, в основі якого лежить flash-пам’ять (та такого, що не має власного контролеру FTL (Flash Translation Layer)).

2.3. Власне файлове сховище

Власне файлове сховище — це ділянка сховища, що займає більшу частину накопичувача, призначена для зберігання файлів. Адресація всередині сховища виконується за блоками зазначеного розміру (4 КіБ) таким чином що, якщо допустити, що ділянка власне сховища починається за байтовою адресою `0x1000000`, то внутрішня адреса `0x10` буде відповідати байтовій адресі `0x1002000`. Таким чином досягається зменшення необхідного для адрес об’єму накопичувача та/або збільшення можливого загального об’єму тег-орієнтованого файлового сховища при збереженні певної довжини адреси.

Кожен файл внутрішньо в рамках сховища визначається як послідовність блоків, що він займає. Останній блок файлу зазвичай є зайнятим не повністю (так як файл може мати будь-який розмір, не обов’язково кратний розміру блока); цей незайнятий простір останнього блока може бути використано для хвостового пакування файлів, якщо користувач виконав побудову сховища з підтримкою хвостового пакування.

Механізм хвостового пакування дозволяє адресувати менші, ніж один блок, об’єми інформації в рамках блоку, а відповідно — зберігати в одному блоці інформацію, що належить до кількох різних файлів. “Хвостовим” це пакування називається тому, що лише останній (“хвостовий”) блок файлу може бути не зайнятий цілком — відповідно, декілька таких блоків спаковуються в один, щоб зменшити об’єм використаного для сховища простору накопичувача.

В ідеальному випадку один файл розміщується на накопичувачі як єдина послідовність блоків, проте можливе також розміщення файлу у вигляді кількох послідовностей — так звана фрагментація. У випадку фрагментації файлу його необхідно читати чи записувати в фізично рознесених позиціях накопичувача, а відповідно — зберігати адреси всіх цих позицій. Для роботи з фрагментованими файлами використовується таблиця управління фрагментацією.

Видалені файли не видаляються з накопичувача фізично, а лише помічаються, як видалені, в таблицях мета-даних. Таким чином забезпечується можливість відмінити видалення (якнайменше) найостаннішого видаленого файлу за умови відсутності записів з моменту видалення. Загалом, відміна видалення лише останнього файлу та лише за умови відсутності запису є виродженим крайовим випадком (повністю заповнене сховище); за умови наявності вільного простору сховища для відновлення буде доступний більший об’єм видалених файлів.

Допоки на накопичувачі залишається незайнятий простір, нові файли розміщуються послідовно; адреса позиції лише збільшується. Таким чином забезпечується можливість відновлення якомога більшого числа видалених файлів, а також відсутність фрагментації. Дописування нової інформації в файли не підтримується сховищем на рівні накопичувача; будь-яка зміна призводить до видалення старого файлу та створення нового.

Проте дописування підтримується на рівні інтерфейсу сховища. Сховище не виконує виділення блоків та запис на накопичувач якомога довше, зберігаючи файлові дані в пам’яті системи, поки не буде вичерпано заданий час невідкладного запису (за замовчуванням пропонується значення 5 секунд, проте воно може бути збільшене для підвищення продуктивності сховища чи меншої кількості записів у випадку накопичувача на базі flash-пам’яті чи твердотілого накопичувача, або зменшене для підвищення безпеки даних — адже у випадку раптового завершення роботи всі операції, не виконані на накопичувачі, буде втрачено), виконано системний виклик `sync()` або використано весь доступний об’єм оперативної пам’яті. Навіть коли вичерпується заданий час невідкладного запису, запис файлу виконується до журналу з додатково зарезервованим для дописування простором, якщо цей файл активно дописувався досі, щоб зменшити необхідність частого перезапису у ділянку власне сховища.

Після того, як накопичувач повністю послідовно заповнено один раз, сховище розміщує нові файли, перезаписуючи видалені. Кожний випадок перезапису або повністю перекриває раніше існуючий видалений файл (у випадку, коли новий файл потребує такої ж кількості блоків, як існуючий), або перекриває його частину та виконує видалення записів, що стосуються видаленого файлу, з таблиць мета-даних (також можливе перенесення видаленого файлу до кошика), помічаючи раніше зайнятий видаленим файлом простір як вільний (якщо новий файл потребує меншої кількості блоків, ніж видалений).

2.4. Кошик

Кошик — це ділянка сховища для тимчасового зберігання файлів незначного розміру, видалених чи перезаписаних в процесі використання сховища. Переміщення файлів до кошика виконується на основі оцінки

евристичним алгоритмом заради запобігання фрагментації в майбутньому (тобто переміщуються лише якщо за алгоритмом оцінки об’єднання вільних ділянок до та після згаданого файлу є доречним). Не видалені переміщені до кошика файли повертаються в нову позицію на накопичувачі за першої можливості.

Також кошик виконує функції журналу, за необхідності зберігаючи файлову інформацію, що очікує запису, та інформацію щодо запланованих дій з накопичувачем.

2.5. Розміщення таблиць у пам’яті

Деталі розміщення таблиць у пам’яті накопичувача можуть бути визначені конкретною реалізацією. В рамках еталонної реалізації використовуються послідовні рядки (масиви) структур чітко визначеного формату, що індексуються своїм положенням в пам’яті накопичувача відносно порядку таблиці, масштабованим відповідно до розміру запису (тобто, якщо запис містить 32 байти, то запис за адресою 0x0 відносно початку таблиці має індекс 0, запис за адресою 0x20 має індекс 1 тощо). У всіх випадках, коли таблиця містить дані змінної довжини (наприклад, список тегів) використовується посилання у зону вільної пам’яті. Для представлення конкретних сутностей (наприклад, тегів, файлів тощо) використовується їх індекс у відповідній таблиці.

2.6. Унікальна ідентифікація файлу

Для унікальної ідентифікації файлу використовується хеш-сума його змісту. Це не призводить до будь-яких затримок у обробці, адже для запису на накопичувач файл неминуче потрапляє до оперативної пам’яті, а сам запис виконується в будь-якому випадку поблоково та займає суттєво більший час, ніж обрахування значення хеш-функції від блоку, що записується.

Використання хеш-суми як ідентифікатора, сумісно з неможливістю дописувати збережений на накопичувач файл, дозволяє виконувати дедуплікацію файлів та виконання перевірки файлу на цілісність на рівні файлового сховища. В рамках еталонної реалізації застосовується хеш-функція Keccak (SHA3) з довжиною значення 512 біт (обрана з метою забезпечити якомога вищу унікальність ідентифікатора файлу); для підтримання сумісності між реалізаціями необхідно дотримуватись використання саме наведеної хеш-функції.

2.7. Запобігання фрагментації

Для запобігання фрагментації використовується “розумне” розміщення файлів на накопичувачі. Заради запобігання фрагментації (з огляду також на основні умови використання тег-орієнтованих файлових сховищ) не реалізована можливість дописувати чи редагувати файл “на місці”. Будь-яка зміна файлу в рамках сховища є операцією створення нового файлу та видалення старого. Це призводить до неможливості швидко вносити зміни до файлів значного розміру (тобто, тег-орієнтоване сховище стає непридатним для роботи з, наприклад, файлами текстового логування), проте значно спрощує реалізацію та підвищує ефективність профілактики фрагментації, адже однією з основних причин фрагментації є дописування вже існуючого файлу, коли ділянка після нього не є вільною та доступною для запису до неї.

Також заради запобігання фрагментації перша ітерація заповнення накопичувача проходить “від краю до краю”, так що всі файли розміщуються послідовно і жоден з видалених не перезаписується (за виключенням випадку коли видалений файл має розмір, рівний розміру нового файлу. Коли розміри становлять сотні блоків, таке співпадіння трапляється рідко і є доцільним перезаписати файл, щоб уникнути фрагментації в майбутньому. Але коли розмір файлу обраховується

одиницями блоків, перезаписувати його недоцільно — такі невеликі ділянки вільного місця зазвичай досить легко відшукати навіть на заповненому накопичувачі, і файли такого розміру вимагають фрагментації дуже рідко).

Після завершення першої ітерації заповнення нові файли мають бути записані на місце видалених старих (або у раніше очищене вільне місце). Видалений файл для перезапису новим вибирається за результатом роботи евристичної оцінювальної функції, що враховує розмір файлів та вільних ділянок, час їх видалення та середній розмір файлу у сховищі. Загалом, оцінка видається за такими загальними принципами:

— Значно вища, якщо розмір ділянки точно співпадає з розміром файлу, що перезаписується; вища, якщо розмір ділянки близький до суми розміру файлу, що записується, та середнього розміру файлу у сховищі, або перевищує таку суму. В інших випадках нижча;

— Тим вища, чим давніший час видалення файлу (для вільних ділянок час видалення в рамках оцінювальної функції вважається нульовим), за логарифмічним законом (зі збільшенням різниці часу видалення та нинішнього часу вага одиниці часу для оцінювальної функції зменшується).

Новий файл завжди записується в ділянку з якомога меншою адресою. Якщо файл займає ділянку не повністю, не зайнята частина вважається новою вільною ділянкою. Якщо новий файл записано поверх раніше видаленого, всі записи щодо видаленого файлу видаляються з таблиць мета-даних.

Також для профілактики фрагментації (та автоматичної дефрагментації в процесі використання сховища) використовується ділянка кошика на накопичувачі. Коли оцінювальна функція дає однаково погані оцінки для всіх доступних ділянок (тобто, показує, що запис до будь-якої з них призведе до фрагментації файлу або небажаної фрагментації вільного

простору), файл (в подальшому файл1) зберігається до кошика/журналу і виконується пошук невеликих файлів (в подальшому файл2), видалення яких дозволило б зберегти файл1 без фрагментації або інших небажаних наслідків (тобто, видалення файл2 має створити ділянку, яка буде високо оцінена оцінювальною функцією). Файл2, що отримує найвищу оцінку, переноситься до кошика, файл1 записується до створеної високо оціненої ділянки, і виконується пошук нової позиції на диску вже для файл2.

Може бути активовано також процедуру періодичної автоматичної дефрагментації. Ця процедура виконується автоматично, за заданим періодом часу. У випадку наявності у сховищі фрагментованих файлів процедура дефрагментації виконує пошук нової ділянки для них, що була б оцінена високо, і якщо така ділянка буде знайдена — переміщує файл до неї. Також процедура автоматичної дефрагментації може виконувати дефрагментацію вільного простору (а саме, переміщення файлів на накопичувачі таким чином, щоб весь вільний простір мав вигляд однієї послідовної ділянки).

Запобігання дефрагментації за допомогою кошика та особливо автоматична дефрагментація не є швидкими процесами, проте достатньо ефективно вирішують проблеми, поставлені перед ними. В більшості випадків використання тег-орієнтованого файлового сховища, побудованого на накопичувачі з використанням прямого доступу, інші механізми запобігання фрагментації ефективно позбавляють сховище необхідності вдаватись до цих часозатратних процедур. Очікувана рідкість їх використання робить прийнятними значні затрати часу.

Механізми запобігання фрагментації активуються лише за умови використання типів накопичувачів, що вимагають пошуку фізичного блоку шляхом обертання, прокручування тощо. У випадку використання накопичувача, що використовує технологію flash-пам’яті, фрагментація не

є значною проблемою. Натомість, необхідно враховувати кількість записів до кожного блоку, адже ресурс блоку є обмеженим.

Тому в цьому випадку використовуються простіші оцінювальні функції (лише щоб мінімізувати використання мета-даних для зберігання інформації про фрагментацію). Також рекомендується встановлення довшого часу невідкладного запису для роботи зі сховищем.

2.8. Оптимізація за ресурсом запису фізичного блока

Основною оптимізацією тег-орієнтованого файлового сховища, побудованого на накопичувачі, в основі якого лежить flash-пам’ять (та такого, що не має власного контролеру FTL) є оптимізація за ресурсом запису фізичного блока. При побудові сховища на такому накопичувачі створюється спеціальна таблиця обліку записів, що відслідковує кількість записів у кожен блок. За можливості, записи виконуються до блоків, що мають найменші значення кількості записів в цій таблиці. Для досягнення цієї мети ніколи не виконується фрагментація файлів, проте може виконуватись переміщення чи зсув файлів в межах накопичувача. Відмова від можливості дописування чи редагування файлів “на місці” спрощує цей процес, дозволяючи більш часті переміщення файлів в рамках нормальної роботи користувача зі сховищем, без додаткових затрат часу.

Окремою складною задачею оптимізації за ресурсом запису фізичного блока є задача розміщення таблиць мета-даних та журналу — ділянок накопичувача, запис в які виконується найчастіше. В рамках запропонованого методу розглядаються лише такі способи вирішення цієї задачі, як буферизація записів метаданих (тобто якомога довше зберігання даних в пам’яті перед записом на накопичувач, в тому числі проміжний запис до журналу якщо ресурс запису журналу це дозволяє, заради мінімізації записів в ділянку мета-даних) та переміщення ділянок мета-даних та журналу в рамках накопичувача, коли ресурс запису блоків

ділянки мета-даних досягає критичного значення. Переміщення ділянок журналу та метаданих стає можливим за умови використання так званого master-блока (адресно — найпершого блока накопичувача), в якому вказуються адреси початку ділянок мета-даних та журналу та їх розмір. Master-блок перезаписується якомога рідше (переміщення ділянок мета-даних та журналу є, взагалі, досить затратною за часом процедурою), завдяки чому гарантується довгий строк його служби навіть за умови використання одного й того ж самого фізичного блоку. Переміщення ділянок мета-даних та журналу призводить до фрагментації ділянки власне файлового сховища, що дещо ускладнює обрахунок адрес, проте є доцільним для підвищення строку служби накопичувача.

В рамках подальшого розвитку методу побудови тег-орієнтованого файлового сховища на накопичувачі з використанням прямого доступу є доцільним пошук інших вирішень для цієї задачі. Наприклад, можливим вирішенням є фрагментація ділянок мета-даних та журналу, проте така фрагментація значно ускладнює роботу з мета-даними та журналом і неможливо сказати без проведення дослідження, чи буде взагалі доцільним таке вирішення.

2.9. Адаптивна природа оптимізації за ресурсом запису та запобігання фрагментації

Всі описані оптимізації щодо ресурсу запису блоку не є доцільними для дискових накопичувачів (що взагалі не мають чітко визначеного ресурсу запису та виходять з ладу зазвичай через механічні несправності) та накопичувачів, в основі яких лежить flash-пам’ять, але які мають FTL-контролер (оптимізацію за записами та рівномірне розповсюдження записів по накопичувачу виконує в такому випадку сам FTL-контролер). У останньому випадку доцільно лише збільшення часу невідкладного запису.

При використанні твердотілих та інших накопичувачів, в основі яких лежить flash-пам’ять, (так як профілактика фрагментації не є критичною для них) ділянка кошику використовується лише для журналювання (та лише коли журнал ввімкнено користувачем).

2.10. Пакування кінцевих частин файлу

Використання хвостового пакування файлів в тег-орієнтованому файловому сховищі є опціональним та може бути ввімкнуто чи вимкнуто на етапі створення сховища на накопичувачі. Для підтримки хвостового пакування створюється додаткова таблиця мета-даних для хвостового пакування, що використовує ідентичні з таблицею адресної індексації індекси (або розширюється таблиця адресної індексації). Якщо в таблиці адресної індексації зберігається лише інформація щодо першого блоку файлу, його розміру в блоках (не враховуючи останній блок) та розміру зайнятої частини останнього блоку в байтах, то в таблиці мета-даних для хвостового пакування додатково зберігається адреса останнього блоку (так як цей блок за умови використання хвостового пакування може бути розміщений окремо від всіх інших блоків файлу), адреса початку інформації, що належить до даного файлу, в рамках останнього блоку, та мітка-“прапорець”, що вказує, чи є даний блок вже запакованим.

За умови використання хвостового пакування, при записі файлу на диск (додатково до всіх інших описаних процедур) виконується пошук частково вільного блоку (хвостового блоку іншого файлу), вільний простір якого якнайбільш точно відповідав би об’єму “зайвої” інформації нового файлу. Якщо такий блок вдається знайти (пошук виконується з допуском невеликого відхилення, порядку десятків байтів), “зайва” хвостова інформація нового файлу записується до нього та виконуються відповідні записи до таблиці мета-даних для хвостового пакування.

Мітка-“прапорець” використовується для пришвидшення пошуку придатного блоку. Вона не встановлена лише коли в блок записано частину лише одного файлу (тобто інша частина блоку вільна — блок доступний для хвостового пакування) і встановлюється (для обох файлів) коли в блок записано частини двох файлів одночасно. Тому при пошуку блоку, придатного для пакування в нього хвостової частини нового файлу, встановлена мітка дозволяє швидко пропустити блок, не виконуючи обрахунок вільного простору в ньому та порівняння з необхідним.

При хвостовому пакуванні частини файлу записуються до блоку не послідовно. При першому записі до блоку інформація поміщається за найнижчою в рамках блоку (нульовою) адресою. При другому записі (пакуванні) інформація записується за найвищою можливою в рамках блоку адресою (таким чином, щоб в блоці не залишилось вільного місця за старшими, ніж останній байт інформації, адресами). Якщо розглядати блок як геометрично лінійний рядок інформації, можна сказати, що він, будучи запакованим, містить в собі інформацію на початку та на кінці, тоді як середина може бути вільною. Саме такий підхід виконується для того, щоб після видалення першого файлу (частина якого була записана в початок блоку) можна було запакувати в початок блоку новий файл, не допускаючи при цьому фрагментації вільного простору всередині блоку.

В запропонованому методі дозволяється пакування хвостових частин лише двох файлів в один блок. Таким чином спрощується управління хвостовим пакуванням, обрахунок адрес та профілактика фрагментації вільного простору всередині блока. Наприклад, при такому підході можливо використовувати всього один біт-“прапорець” в таблиці як мітку доступності блока для хвостового пакування.

2.11. Журнал

Журнал (для журналу використовується ділянка кошика, частина якої виділяється саме під потреби журналювання) також може бути ввімкнено чи вимкнено користувачем. Вимкнення журналу можливо виконати як тимчасово (ділянка простору накопичувача, виділена для журналу, буде в такому випадку не використовуватись) так і назавжди (при побудові сховища на накопичувачі — ділянка простору під журнал виділятись не буде).

Журнал використовується для швидкого збереження ключової інформації на накопичувачі, повний запис якої може вимагати значного часу, та для забезпечення цілісності даних у випадку раптового завершення роботи системи. Для підвищення швидкодії сховища рекомендується використовувати журналювання лише для мета-даних, проте можливе повне та часткове журналювання і файлових даних.

Дані в журналі зберігаються у вигляді послідовних записів. Кожен запис має часову мітку та поміщається в журнал в циклічному порядку (тобто, записи спочатку послідовно займають весь виділений під журнал простір, а потім кожен новий запис поміщається на місце найстарішого). Таким чином, у випадку раптового завершення роботи за записами в журналі можливо повністю відновити стан сховища на момент згаданого раптового завершення роботи (за винятком останніх операцій, що не були записані до журналу). Гарантується цілісність сховища та стійкість проти раптового завершення роботи (адже пошкодженим через перерву в процесі запису може бути, якнайгірше, лише один запис в журналі).

Першою перевагою журналювання є можливість швидкого запису інформації щодо спланованих дій, які ще не узгоджено остаточно на момент такого запису. Наприклад, якщо в процесі запису файлу операційною системою вичерпується час невідкладного запису, але є підстави вважати, що запис файлу ОС не завершено, відомі на момент

вичерпання часу невідкладного запису дані можуть бути записані до журналу. Потім з кількох таких записів буде утворено єдиний файл, і таким чином уникнуто необхідності виконувати більш, ніж одне, оновлення таблиць мета-даних (та, можливо, переміщення файлу у випадку, якщо знайдена для його першої відомої частини ділянка не здатна вмістити файл цілком).

Другою (та основною) перевагою журналювання є забезпечення цілісності даних та сховища. Заплановані за записами в журналі операції виконуються на інших ділянках у фоновому режимі. Навіть якщо роботу сховища перервати в момент такого запису, при наступному запуску буде можливо відшукати момент перерви за записами в журналі та повторити операцію, перезаписуючи пошкоджені дані коректними. Якщо ж через раптову перерву роботи відбувається пошкодження запису в самому журналі, такий запис видаляється (і буде, в найгіршому випадку, єдиною втраченою одиницею інформації). Саме через можливість гарантувати цілісність даних рекомендується обов’язково використовувати журналювання для мета-даних: незважаючи на те, що втрата файлу може призвести до значних збитків, в більшості випадків швидкодія є більш значущим фактором; проте пошкодження мета-даних може призвести до непрацездатності всього сховища та втрати повного об’єму збережених до нього файлів.

Незважаючи на значущість обох наведених переваг, журналювання має недоліки: сповільнення роботи сховища (особливо якщо використовується журналювання даних) та (при невеликому часі невідкладного запису) частіші записи до накопичувача, що скорочують строк служби у випадку використання накопичувачів, що використовують flash-пам’ять. Також журнал використовує додатковий простір накопичувача. Якщо оптимізація за цими напрямками є більш доцільною,

ніж збереження цілісності накопичувача, журналювання може бути вимкнуто.

Розмір журналу можливо задати на етапі побудови сховища на накопичувачі. Більший розмір журналу дозволяє кращу оптимізацію записів в основну частину диску, проте залишає менше простору для власне файлів. Також розмір журналу доцільно задавати тим більшим, чим більшим задається час невідкладного запису; інакше можлива ситуація, коли за час невідкладного запису в пам’яті системи накопичується більше операцій, ніж здатен вмістити журнал (особливо актуально у випадку, коли використовується журналювання даних). При раптовій зупинці роботи сховища в такій ситуації може бути втрачено більш, ніж один журнальний запис, операцій, адже частина записів залишиться в пам’яті системи в очікуванні обробки вже зроблених в журнал записів.

2.12. Забезпечення цілісності

Для перевірки цілісності прочитаних з накопичувача даних широко використовуються значення хеш-функцій та контрольні суми.

Зі вмісту кожного збереженого в сховище файлу при записі обраховується значення хеш-суми. Також для кожної з таблиць, для записів в деяких таблицях (залежить від реалізації), та для записів в журналі обраховується та зберігається контрольна сума.

При читанні файлу, записів з таблиць чи записів з журналу виконується повторне обрахування хеш-суми чи контрольної суми. Порівняння повторно обрахованого значення зі збереженим дозволяє визначити, чи читання було виконано без помилок. Якщо спостерігається помилка (нерівність значень) навіть після кількох повторних помилок читання, можна зробити висновок щодо помилки запису та спробувати (за наявності такого запису в журналі) відновити файл/запис таблиці за журналом або повідомити користувача про помилку та втрату інформації.

Обраховані значення хеш-суми від вмісту також використовуються в якості ідентифікаторів файлів. Таким чином забезпечується дедуплікація: при спробі запису файлу з уже відомою хеш-сумою від вмісту, новий файл не буде створено. Натомість, задані для нього теги буде присвоєно до вже наявного у сховищі файлу.

3. СТРУКТУРИ ДАНИХ І АЛГОРИТМИ ПРОГРАМНОЇ СИСТЕМИ

В даному розділі наводиться опис еталонної реалізації запропонованого методу побудови тег-орієнтованого файлового сховища на накопичувачі з використанням прямого доступу, без використання проміжної ієрархічної файлової системи.

В рамках усіх структур даних, що використовуються файловим сховищем, багатобайтні числа зберігаються в порядку “big endian” (від старшого до молодшого). Нумерація бітів в рамках описання формату виконується в порядку в порядку “MSB 0” (найстарший біт нумерується, як нульовий).

Описані структури даних часто використовують так зване “магічне число” для забезпечення цілісності даних. “Магічне число” — це константа, визначена для певної структури даних як частина її формату, що залишається завжди незмінною. Обов’язковою частиною процесу зчитування даних має бути перевірка “магічного числа”. Не співпадіння даних, прочитаних з пам’яті накопичувача, з очікуваною константою є ознакою помилки формування структур сховища на накопичувачі або їх пошкодження.

Також “магічне число” може слугувати для ідентифікації початку наступного блоку, так як розміщується на першій позиції в більшості випадків.

3.1. Master-блок

На початку першого доступного блоку накопичувача розміщується так званий master-блок. Master-блок містить в собі адреси початку кожної з трьох логічних ділянок сховища (мета-дані, власне файлове сховище та кошик/журнал) та їх розміри, а також адреси кожної з таблиць метаданих

(та їх розміри) і адресу журналу (необхідна в тому випадку, коли використовуються і журнал, і кошик) та його розмір. Розмір ділянки власне файлового сховища не вказується, адже ця ділянка за визначенням займає весь простір накопичувача, не зайнятий іншими ділянками. В рамках еталонної реалізації для всіх трьох ділянок не допускається фрагментація (розбиття ділянки на декілька частин так, що між частинами ділянки розміщується інша ділянка чи її частина).

Master-блок завжди розміщується за фіксованою адресою (першою доступною, що в подальшому буде розглядатися як “нульова адреса”, або “адреса 0”), тому (з огляду на особливості роботи накопичувачів, побудованих на основі flash-пам’яті) має перезаписуватись якомога рідше, щоб не викликати швидкого зношення блока, що включає в себе нульову адресу.

Мінімізація перезаписів master-блока є природною та очевидною, адже сама інформація, яку він в собі містить — адреси ключових структур даних — не є такою, що повинна змінюватись часто в процесі експлуатації. Саме тому еталонна реалізація в принципі не допускає можливості перезапису master-блока. Підтримка переміщення ділянок, журналу чи окремих таблиць (необхідна, як описано у попередньому розділі, для більш повного та рівномірного використання накопичувачів, побудованих на основі flash-пам’яті), що потребувала б можливості перезапису master-блока, може бути реалізована в рамках подальшого розвитку метода для адаптивної роботи з різними типами накопичувачів.

Еталонна реалізація допускає формування master-блока лише один раз, при створенні сховища (форматуванні накопичувача). На цьому етапі користувачу дозволяється обрати окремі параметри роботи сховища, наприклад, ввімкнути чи вимкнути хвостове пакування файлів чи задати розмір журналу.

Формат master-блока, що використано в еталонній реалізації, описує табл. 3.1. Значення колонки “Розмір” вказано в бітах. Значення колонки “Відносна адреса” вказано в байтах від початку master-блока. Всі розміри ділянок записуються в блоках. Всі адреси записуються в блоках як абсолютні значення (від початку накопичувача). Всі поля є обов’язковими. Поля, що описують не наявні фактично структури (таблиці, журнал тощо), обнуляються (нульова адреса чи розмір не є дозволеними значеннями і розглядаються сховищем як “false”).

Таблиця 3.1 — Формат master-блока

Відносна адреса	Розмір	Призначення
`0x00`	16	“Магічне число”: `0x9cd6`
`0x02`	16	Мітки
`0x04`	32	Адреса ділянки мета-даних
`0x08`	24	Розмір ділянки мета-даних
`0x0b`	32	Адреса ділянки власне файлового сховища
`0x0f`	32	Адреса ділянки кошика
`0x13`	24	Розмір ділянки кошика
`0x16`	32	Адреса журналу
`0x1a`	16	Розмір журналу

Поле “мітки” містить 16 бітів-“прапорців”, що задають важливі опції щодо функціонування файлового сховища. Структура цього поля наведена в таблиці 3.2.

Таблиця 3.2 — Структура поля “мітки” master-блока

Біт	Призначення
0	Мітка використання журналу
1	Мітка пакування кінцевих частин файлів
2-15	Зарезервовані біти

3.2. Заголовок ділянки мета-даних

Ділянка мета-даних має власний заголовок. Формат заголовку ділянки мета-даних наведено у таблиці 3.3. Ділянки власне файлового сховища та кошика не мають власного заголовку, адже вся інформація, необхідна для роботи з ними, надається в рамках master-блока та таблиць ділянки мета-даних. Читати таблицю формату заголовку ділянки мета-даних необхідно так само, як і таблицю формату master-блока (розміри вказано в бітах, адреси в байтах від початку заголовка тощо).

Таблиця 3.3 — Формат заголовку ділянки мета-даних

Відносна адреса	Розмір	Призначення
`0x00`	32	“Магічне число”: `0x4498d482`
`0x04`	32	Адреса таблиці адресації
`0x08`	16	Розмір таблиці адресації
`0x0a`	32	Адреса таблиці файлових мета-даних
`0x0e`	16	Розмір таблиці файлових мета-даних
`0x10`	32	Адреса таблиці тегових мета-даних
`0x14`	16	Розмір таблиці тегових мета-даних
`0x16`	32	Адреса таблиці індексації тегами
`0x1a`	16	Розмір таблиці індексації тегами
`0x1c`	32	Адреса таблиці індексації файлами
`0x20`	16	Розмір таблиці індексації файлами
`0x22`	32	Адреса таблиці індексації незайнятого простору накопичувача
`0x26`	16	Розмір таблиці індексації незайнятого простору накопичувача
`0x28`	32	Адреса таблиці видалених файлів
`0x2c`	16	Розмір таблиці видалених файлів
`0x2e`	32	Адреса таблиці управління фрагментацією
`0x32`	16	Розмір таблиці управління фрагментацією

0x34	32	Адреса таблиці мета-даних для хвостового пакування
0x38	16	Розмір таблиці мета-даних для хвостового пакування
0x3a	32	Адреса таблиці обліку записів
0x3e	16	Розмір таблиці обліку записів
0x40	32	Кількість вільних записів в таблиці адресації

На відміну від master-блока, заголовок ділянки мета-даних допускає як завгодно часті перезаписи та не є незмінним. Наприклад, можливе переміщення таблиць всередині ділянки мета-даних для рівномірного розповсюдження записів за фізичними блоками накопичувача (в рамках оптимізації строку служби накопичувачів, побудованих з використанням flash-пам’яті), а також зміна розмірів таблиць. Еталонна реалізація не надає змоги виконувати ці дії, проте їх підтримка є вбудованою в управляючі структури даних для простого розширення функціональності в подальшому без необхідності переформатування накопичувача.

Заголовок ділянки мета-даних також може переміщуватись (у тому випадку, коли блок, що містить заголовок, досягає небезпечної кількості перезаписів). Якщо здійснюється переміщення заголовку ділянки мета-даних, в перший блок ділянки записується вказівник на його нове положення. Формат такого вказівника наведено в таблиці 3.4.

Таблиця 3.4 — Формат заголовка-вказівника на перенесений заголовок

Відносна адреса	Розмір	Призначення
0x00	32	“Магічне число”: 0x4498d482
0x04	32	Число “0”
0x08	32	Адреса переміщеного заголовка

Однозначно відрізнити повноцінний заголовок ділянки мета-даних від вказівника на переміщений заголовок дозволяє перегляд другого поля прочитаної з накопичувача структури даних. Поле має однаковий розмір в обох випадках, проте ніколи не може бути рівним 0 у повноцінному заголовку: по-перше, за нульовою адресою на накопичувачі завжди розміщується master-блок, і ця адреса не є допустимою як адреса таблиці адресації; по-друге, на відміну від деяких інших таблиць (наприклад, таблиці мета-даних для хвостового пакування), таблиця адресації має бути обов’язково присутня в будь-якому коректно побудованому за запропонованим методом сховищі, і таким чином у випадку таблиці адресації нульова адреса не може означати “таблиця відсутня”.

Отже, єдиним коректним випадком, коли в першому блоці ділянки мета-даних друге 32-бітне поле буде повністю обнуленим, може бути той випадок, коли заголовок було переміщено. Необхідно прочитати наступне 32-бітне поле і шукати заголовок за вказаною в ньому адресою.

В загальному випадку, вказівник може вказувати не на сам заголовок, а на інший вказівник (якщо перший блок ділянки мета-даних повністю вичерпав ресурс даних і перемкнутий контролером в режим “лише читання”, це — єдиний спосіб перемістити вже переміщений раніше заголовок в нове місце), і так далі, проте подібний ланцюг вказівників знижує продуктивність, ускладнює роботу зі сховищем і загалом не є доцільним, тому за нормальних умов роботи не допускається.

3.3. Заголовок журналу

Також власний заголовок має журнал. Формат заголовку журналу наведено в таблиці 3.5.

Загалом, рекомендується вказувати розмір блоку журналу таким самим, як і основний в рамках тег-орієнтованого файлового сховища розмір блоку. Проте можливо вказання як меншого (у випадках, коли

Таблиця 3.5 — Формат заголовку журналу

Відносна адреса	Розмір	Призначення
`0x00`	32	“Магічне число”: `0x80ce3c46`
`0x04`	32	Індикатор типу блока (`0xffffffff`)
`0x08`	32	Розмір блоку журналу (в байтах)
`0x0c`	32	Максимальна кількість блоків у журналі
`0x10`	32	Відносна адреса (в байтах) початку робочої секції журналу
`0x14`	32	Розмір робочої секції журналу (в байтах)
`0x18`	32	Адреса останнього виконаного запису

використовується лише запис мета-даних через журнал), так і більшого (у випадках, коли використовується запис файлів через журнал та відомо, що файли зазвичай мають більший за заданий основний розмір блоку, розмір). Використання меншого розміру блоку в коректній ситуації дозволяє помістити більше записів у журнал того самого розміру, тоді як використання більшого розміру блоку дозволяє пришвидшити роботу з журналом завдяки використанню меншої кількості записів.

Але використання невідповідного розміру блоку (наприклад, меншого за очікуваний розмір файлу при записі файлів через журнал, або більшого за очікуваний розмір мета-даних при записі лише мета-даних) призводить до негативних наслідків. В першому випадку (занадто малий розмір блоку) робота сховища буде сповільнена, адже для запису в журнал однієї дії буде необхідно використовувати декілька блоків. В другому випадку (занадто великий розмір блоку) буде використано дарма багато вільного простору накопичувача.

Також при використанні розміру блоку журналу, меншого за фізичний розмір блоку накопичувача, збільшується ризик втрати даних, адже для нового запису в журнал буде необхідно перезаписати блок (що містить кілька записів за такої умови) цілком. Відповідно, якщо робота системи

буде раптово припинена під час такого перезапису, будуть втрачені всі журнальні записи цього блоку.

Іншим недоліком використання відмінного від фізичного розміру блоку накопичувача розміру блоку журналу є сповільнення роботи сховища у випадку, коли блок журналу розміщується на декількох блоках накопичувача одночасно. Для запису одного журнального блоку в такому випадку необхідно записати два чи більше блоки накопичувача, витрачаючи відповідно більше часу.

3.4. Таблиця адресації

Таблиця адресації вміщує в собі всі основні дані щодо розміщення файлових даних на диску, а також слугує основною таблицею файлової індексації (саме індекс у таблиці адресації використовується в інших таблицях для визначення, про який саме файл йде мова). Індексація у таблиці адресації виконується за положенням запису в ній (тобто, індекс першого запису — 0, другого запису — 1, і так далі). Це є можливим завдяки тому, що кожен запис в таблиці індексації має фіксований, відомий наперед розмір.

Формат запису для таблиці адресації наведено в таблиці 3.6. Важливою особливістю даної таблиці є те, що в рамках еталонної реалізації таблиця адресації є суміщеною з таблицею мета-даних для хвостового пакування. Таким чином, в таблиці можуть бути наявні чи відсутні деякі поля в залежності від налаштувань конкретного сховища.

В таблиці 3.6 наведено формат для випадку, коли пакування використовується. В таблиці 3.7 — для випадку, коли пакування не використовується.

Як можна спостерігати, у випадку використання пакування в кожному записі таблиці зберігаються додаткові специфічні для випадку пакування дані: адреса останнього блоку файлу (без використання пакування ця

Таблиця 3.6 — Формат запису для таблиці адресації (з пакуванням)

Відносна адреса	Розмір	Призначення
`0x00`	16	“Магічне число”: `0xcef7`
`0x02`	32	Адреса першого блоку файлу
`0x06`	32	Розмір файлу в блоках
`0x0a`	12	Розмір зайнятої частини останнього блоку в байтах
`0x0b + 4bit`	12	Адреса частини файлу в останньому блоці
`0x0d`	32	Адреса останнього блоку файлу
`0x11`	8	Мітки
`0x12`	16	Контрольна сума crc16

Таблиця 3.7 — Формат запису для таблиці адресації (без пакування)

Відносна адреса	Розмір	Призначення
`0x00`	16	“Магічне число”: `0xcef7`
`0x02`	32	Адреса першого блоку файлу
`0x06`	32	Розмір файлу в блоках
`0x0a`	12	Розмір зайнятої частини останнього блоку в байтах
`0x0b + 4bit`	4	“0” (всі чотири біти встановлено в “0” для вирівнювання запису за байтами)
`0x0c`	8	Мітки
`0x0d`	16	Контрольна сума crc16

адреса, очевидно, обраховується як “адреса першого блоку + розмір файлу в блоках”) та адреса частини файлу в останньому блоці (без використання пакування, очевидно, рівна 0). Докладніше механізм пакування кінцевих частин файлу (або хвостового пакування) описаний у відповідному підрозділі.

Поле розміру зайнятої частини останнього блоку, натомість, присутнє завжди, і є необхідним для визначення розміру файлу в байтах (а саме, як

“кількість блоків × розмір блоку + розмір зайнятої частини останнього блоку + 1”). Останній блок ніколи не може бути пустим (в цьому розмір файлу в блоках просто буде меншим на 1), тому значення зайнятої частини останнього блоку збільшується на одиницю (значення “0” означає 1 байт, значення “1” означає 2 байти тощо, і значення “4095” означає 4096 байт — повністю зайнятий блок).

Поле розміру файлу в блоках не включає в себе останній блок (тобто, якщо файл займає лише один блок, значення цього поля буде “0”). У випадку, коли файл є пустим (має розмір 0 байт), поля розміру файлу в блоках та розміру зайнятої частини останнього блоку обидва встановлюються в “0” (так само, як для файлу розміром 1 байт), і також встановлюється мітка пустоти. Загалом, лише встановлення мітки пустоти достатньо (поля розміру ігноруються в цьому випадку).

Поле “мітки” містить 8 бітів-“прапорців”, що вказують на певні особливості стану файлу. Структура цього поля наведена в таблиці 3.8.

Таблиця 3.8 — Структура поля “мітки” запису для таблиці адресації

Біт	Призначення
0	Мітка пустоти (файл є пустим, має розмір 0 байт, і фактично не існує)
1	Мітка видаленого стану (1, якщо файл видалений)
2	Мітка фрагментації (1, якщо файл фрагментований)
3-7	Зарезервовані біти

Використання мітки пустоти дозволяє мінімальне використання простору накопичувача для файлів розміром 0 байт.

Використання мітки видаленого стану в таблиці адресації дозволяє еталонній реалізації сховища уникнути необхідності використання таблиці видалених файлів. Всі дані видалених файлів зберігаються в тих самих

таблицях, що й дані актуальних файлів. Єдиним, що відрізняє видалений файл від актуального, є дана мітка.

Використання мітки фрагментації дозволяє зменшити розмір таблиці управління фрагментацією у випадку, коли накопичувач містить невелику кількість фрагментованих файлів (заради чого використовуються механізми запобігання фрагментації). Звертання до таблиці фрагментації виконується лише коли відповідна мітка встановлена; в інакшому випадку звертання не відбувається і резервувати місце для файлу в таблиці фрагментації не потрібно. У випадку, коли мітка фрагментації встановлена, поле “адреса першого блоку файлу” отримує нове значення: індекс у таблиці фрагментації.

3.6. Таблиця файлових мета-даних

Таблиця файлових мета-даних вміщує в собі всі мета-дані, що стосуються конкретного файлу, але не є при цьому ні теговими даними, ні основними даними, що визначають його розміщення на диску (такими як розмір, адреса першого блоку тощо), наприклад час створення та останнього доступу, Unix-дозволи для власника та групи, значення хеш-функції від вмісту файлу. Індексація у таблиці файлових мета-даних виконується так само, як у таблиці адресації — за положенням запису у таблиці. Це є можливим завдяки тому, що кожен запис в таблиці файлових мета-даних має фіксований, відомий наперед розмір.

Таблиця файлових мета-даних завжди містить стільки ж записів, скільки й таблиця адресації. Записи з однаковими індексами в обох цих таблицях завжди стосуються одного й того ж самого файлу.

Формат запису для таблиці файлових мета-даних наведено в таблиці 3.9.

Таблиця 3.9 — Формат запису для таблиці файлових мета-даних

Відносна адреса	Розмір	Призначення
`0x00`	16	“Магічне число”: `0x57a1`
`0x02`	16	Мітки Unix-властивостей
`0x04`	32	UID власника
`0x08`	32	GID
`0x0c`	64	Час останнього доступу
`0x14`	64	Час останньої зміни
`0x1c`	64	Час останньої модифікації вмісту
`0x24`	64	Час створення
`0x2c`	512	Значення хеш-функції від вмісту файлу
`0x6c`	32	Контрольна сума crc32

Поле “мітки Unix-властивостей” містить 16 бітів-“прапорців”, що відповідають зазначеним в стандарті POSIX властивостям файлів. Структура цього поля наведена в таблиці 3.10.

Таблиця 3.10 — Структура поля “мітки Unix-властивостей”

Біт	Призначення
0-3	Чотири старші біти визначають тип файлу: `0x1` — FIFO. `0x2` — символьний пристрій. `0x4` — збережений запит (в рамках POSIX це тип “директорія”, проте в рамках тег-орієнтованого файлового сховища поняття “директорії” не має ніякого сенсу). `0x6` — блоковий пристрій. `0x8` — звичайний файл. `0xa` — символьне (“м’яке”) посилання. `0xc` — сокет.
4	Мітка S_ISUID (встановити UID під час виконання)

5	Мітка S_ISGID (встановити GID під час виконання)
6	Мітка S_ISVTX (“sticky bit”, використовується для кешування даних файлу в пам’яті)
7	Мітка S_IRUSR (дозвіл читання для власника)
8	Мітка S_IWUSR (дозвіл запису для власника)
9	Мітка S_IXUSR (дозвіл виконання для власника)
10	Мітка S_IRGRP (дозвіл читання для групи)
11	Мітка S_IWGRP (дозвіл запису для групи)
12	Мітка S_IXGRP (дозвіл виконання для групи)
13	Мітка S_IROTH (дозвіл читання для всіх інших)
14	Мітка S_IWOTH (дозвіл запису для всіх інших)
15	Мітка S_IXOTH (дозвіл виконання для всіх інших)

Поля UID власника та GID містять, відповідно, ідентифікатор користувача (User Identifier) та ідентифікатор групи (Group Identifier), для яких в полі “мітки Unix-властивостей” задаються дозволи.

Три типи дозволів (на читання, на запис і видалення та на виконання) задаються окремо для користувача-власника (чий UID записується в поле UID власника), для певної групи користувачів (чий GID також записується у відповідне поле; важливо помітити, що власник не обов’язково має належати до цієї ж групи) та для всіх інших. Такий набір дозволів для файлу склався історично та є достатнім в абсолютній більшості випадків (так як зазвичай в рамках Unix користувач може входити до декількох груп одразу, при необхідності для певного файлу може бути створена цілком окрема група доступу).

32-бітні значення UID та GID дозволяють підтримку систем зі значною кількістю користувачів.

Недоліком визначення дозволів за UID та GID є те, що при перенесенні накопичувача до нової системи немає можливості визначити, що дозволи відносяться до користувачів та груп, на цій системі взагалі не

присутніх. Якщо UID чи GID співпадають з наявними у системі, вважається, що про саме цих користувача та групу йдеться.

Час останнього доступу, або atime (access time) — це кількість секунд з “початку епохи” (опівночі 1 січня 1970-го року за Ґрінвічем), що визначає момент, в який до файлу останній раз здійснювався доступ (будь-який, в тому числі читання). Зазвичай запис часу останнього доступу вмикають або вимикають в операційній системі під час монтування сховища, проте з огляду на відсутність системної інтеграції в рамках запропонованої еталонної реалізації та, відповідно, неможливість монтування, пропонується обрати, чи буде виконуватись запис часу останнього доступу, при побудові сховища. Запис часу останнього доступу може бути доцільно вимкнути, адже при частих читаннях файлів буде постійно виконуватись його оновлення, що може призвести до сповільнення роботи сховища та швидкого витрачення ресурсу перезапису блока (у випадку накопичувачів, побудованих з використанням flash-пам’яті). Але може бути доцільно і вмикати запис часу останнього доступу, адже спостереження за цим параметром файлу є іноді корисним для виявлення несправностей у роботі системи.

Час останньої зміни, або ctime (change time) — це аналогічний часу останнього доступу за властивостями параметр файлу, але відображає не момент останнього здійснення доступу до файлу, а момент останньої зміни файлу. Зміною файлу вважається будь-яка зміна його мета-даних (а відповідно, і будь-яка зміна його вмісту, адже в момент зміни вмісту файлу обраховується нове значення хеш-функції від його вмісту, що зберігається як елемент мета-даних. Як і запис часу останнього доступу, запис часу останньої зміни можна ввімкнути чи вимкнути при побудові тег-орієнтованого файлового сховища на накопичувачі. На відміну від вимикання запису часу останнього доступу, запис часу останньої зміни вимикається рідко, адже запис до таблиці буде виконано при зміні файлу в

будь-якому випадку, за визначенням. Таким чином, вимкнення запису часу останньої зміни файлу не надає ніяких переваг, а лише недоліки.

Час останньої модифікації вмісту, або mtime (modification time) — це аналогічний часу останньої зміни за властивостями параметр файлу, але час останньої модифікації вмісту, на відміну від часу останньої зміни, оновлюється лише коли змінюється вміст файлу та залишається незмінним, коли змінюються мета-дані файлу. Таким чином, розрізняючи час останньої зміни та час останньої модифікації вмісту, можливо прослідковувати як час “будь-якої найостаннішої дії з файлом”, так і час власне змісту самого файлу, а не мета-даних файлового сховища, в якому файл зберігається. Як і запис інших часових полів таблиці файлових мета-даних, запис часу останньої модифікації вмісту можна вимкнути при побудові тег-орієнтованого файлового сховища на накопичувачі. Проте, як і у випадку з часом останньої зміни, це не є доцільним, адже будь-яка зміна вмісту файлу призводить до обрахунку нового значення хеш-функції вій його вмісту і відповідний рядок у таблиці файлових мета-даних буде перезаписано в будь-якому випадку. Таким чином, при вимкненні запису часу останньої модифікації вмісту файлу не можливо отримати ніяких переваг.

Час створення файлу, або crtime (creation time) — це аналогічний іншим часовим параметрам файлу за властивостями параметр, але час створення файлу записується лише один раз — в момент внесення файлу до сховища, тобто в момент запису мета-даних нового файлу в місце таблиці мета-даних, що до цього моменту було не зайнятим. В подальшому час створення файлу не оновлюється. Запис часу створення файлу неможливо вимкнути при побудові тег-орієнтованого файлового сховища на накопичувачі. Також в момент створення нового запису мета-даних файлу параметри часу останнього доступу, часу останньої модифікації

вмісту та часу останньої зміни встановлюються рівними часу створення, навіть якщо їх запис було вимкнуто.

При видаленні файлу час останньої зміни оновлюється, навіть якщо таке оновлення було вимкнуто. В цьому випадку час останньої зміни слугує інформацією про час видалення файлу, що використовується в рамках оцінювальної функції алгоритму запобігання фрагментації для вибору найбільш доцільного для перезапису видаленого файлу. Старші файли отримують вищу оцінку доцільності перезапису, ніж новіші. При відновленні файлу це поле також оновлюється, навіть якщо його оновлення було вимкнуто при побудові тег-орієнтованого файлового сховища на накопичувачі.

Хеш-функція від вмісту файлу обраховується кожного разу, коли файл зчитується з накопичувача чи записується до нього. Порівняння обрахованої функції зі збереженою дозволяє визначити, чи була зміна вмісту файлу, при записі, та чи не було пошкоджено файл на накопичувачі — при зчитуванні. В запропонованій реалізації для хешування вмісту файлу використовується функція SHA3 (Keccak) з розміром значення хеш-функції 512 біт. Загалом, на користь використання саме цієї (чи будь-якої іншої) функції немає значних аргументів, і обрана хеш-функція була довільно. Довжина значення 512 біт забезпечує мінімальну ймовірність повторів значення хеш-функції для різних за вмістом файлів (так званих хеш-колізій), що є важливим для дедуплікації та для використання значення хеш-функції в якості індексу.

3.7. Таблиця тегових мета-даних

Таблиця тегових мета-даних вміщує в собі всі мета-дані, що стосуються конкретного тегу. В рамках запропонованої реалізації ці метадані обмежуються рядком, що визначає тег, та так званими

псевдонімами (збереженими в мета-даних тегу посиланнями на інший тег, з якому даний тег є еквівалентним — тобто є псевдонімом іншого тегу).

Так як рядкове визначення тегу може, в загальному випадку, бути довільної довжини, визначений під таблицю тегових мета-даних простір накопичувача розділяється на дві частини: власне таблиця розміщується на його молодших адресах (“з початку”), а на старших (“з кінця”) розміщуються згадані рядки довільної довжини (кінець рядка визначається нульовим символом). Таким чином, можливо задати фіксований розмір запису в таблиці тегових мета-даних: поле рядкового визначення тегу містить адресу (в байтах) відповідного рядка, адресованого з кінця виділеного під таблицю тегових мета-даних простору.

Отже, виділений під таблицю тегових мета-даних простір на накопичувачі розділяється на дві активні частини, між якими залишається ділянка вільного простору, що дозволяє подальше зростання таблиці. При необхідності збільшити розмір таблиці тегових мета-даних достатньо лише перенести і таблицю, і сховище рядків у нове місце, збільшивши ділянку вільного простору між ними. Так як внутрішня адресація рядків в таблиці тегових мета-даних виконується з кінця виділеного для неї простору, переобраховування адрес буде не потрібно виконувати.

Вважається, що таблиця тегових мета-даних буде обов’язково прочитана з накопичувача в пам’ять системи, що працює зі сховищем, цілком. Наведений формат збереження таблиці на накопичувачі потрібен саме для цього — збереження її між сесіями роботи.

Операція додавання нового тегу вважається достатньо рідкісною, щоб при кожному її виконанні записувати таблицю тегових мета-даних на диск цілком без значних втрат як швидкодії, так і ресурсу запису фізичного блока.

З таблицею тегових мета-даних можна працювати і напряму на накопичувачі, але це призведе до значних затрат часу на лінійний пошук

потрібного тегу за його рядковим визначенням. Для такого пошуку оптимально використовувати хеш-таблиці, проте робота з хеш-таблицями напряму на накопичувачі є недоцільною (ці структури даних мають деяку схильність до частої зміни розміру чи функції хешування, й обидва ці типи змін призводять до необхідності повного переобрахування індексів як у самій таблиці тегових мета-даних, так і у таблицях, що посилаються на неї) і не реалізована в запропонованій еталонній реалізації.

Формат запису для таблиці тегових мета-даних наведено в таблиці 3.11.

Таблиця 3.11 — Формат запису для таблиці тегових мета-даних

Відносна адреса	Розмір	Призначення
`0x00`	16	“Магічне число”: `0xd1be`
`0x02`	32	Адреса рядкового визначення тегу (відносно кінця виділеного для таблиці тегових мета-даних простору, “всередину”, в байтах)
`0x06`	32	Індекс тегу (якщо даний тег є псевдонімом)
`0x0a`	16	Контрольна сума crc16

Рядкові визначення тегів записуються на накопичувач з використанням кодування UTF-8. Єдиним не дозволеним символом є нульовий символ, адже нульовий символ слугує для розділення рядків. Ніяке обмеження на довжину рядкового визначення тегу не накладається (окрім очевидного обмеження — кількості доступного простору накопичувача).

3.8. Перехресні таблиці асоціативної індексації

Перехресні таблиці асоціативної індексації забезпечують можливість швидкого пошуку як тегів, що належать певному файлу, так і файлів, що мають певний тег. Для цього сформовано дві ідентичні за структурою таблиці (таблиця індексації тегами та таблиця індексації файлами), що

мають такі ж самі індекси як, відповідно, таблиця тегових мета-даних та таблиця адресації.

Формат запису для таблиці асоціативної індексації наведено в таблиці 3.12.

Таблиця 3.12 — Формат запису для таблиць асоціативної індексації

Відносна адреса	Розмір	Призначення
`0x00`	16	“Магічне число”: `0x2274`
`0x02`	32	Адреса набору тегів (або файлів)
`0x06`	16	Контрольна сума crc16

Так само, як і у випадку таблиці тегових мета-даних, вміст записів таблиць асоціативної індексації зберігається окремо, у кінцевій частині виділеної для відповідної таблиці ділянки простору накопичувача, адже має змінну довжину. “Набір тегів” (чи “набір файлів”) являє собою масив 32-бітних адрес для, відповідно, таблиці тегових мета-даних чи таблиці адресації. Таким чином, забезпечується можливість призначення одного тегу декільком файлам і декількох тегів — одному файлу.

Недоліком такої організації асоціативної індексації є необхідність частих переміщень наборів тегів та файлів, коли їх розмір змінюється. Негативний вплив цього недоліку зменшується завдяки розміщенню наборів у пам’яті накопичувача з деяким розрахунком на їх збільшення в подальшому (початково резервується 20 байтів на кожний набір — для 5 тегів чи файлів відповідно — а при кожному зростанні понад зарезервований простір резервується в два рази більша частина).

3.9. Таблиця управління фрагментацією

Таблиця управління фрагментацією використовується для того, щоб мати можливість збереження файлу, коли загальний об’єм доступного простору є достатнім для його збереження, проте весь цей простір

розміщено у вигляді фізично не послідовних ділянок (кожна з яких, відповідно, менша за розмір файлу). Коли така необхідність виникає, файл зберігається у вигляді декількох фрагментів. До таблиці адресації записується посилання на таблицю управління фрагментацією, а у таблицю управління фрагментацією — адреси і розмір кожного з фрагментів.

Формат запису для таблиці управління фрагментацією наведено в таблиці 3.13.

Таблиця 3.13 — Формат запису для таблиці управління фрагментацією

Відносна адреса	Розмір	Призначення
`0x00`	16	“Магічне число”: `0x8f12`
`0x02`	32	Адреса фрагменту на накопичувачі
`0x06`	32	Розмір фрагменту в блоках
`0x0a`	32	Індекс наступного фрагменту
`0x0e`	16	Контрольна сума crc16

Поле “Індекс наступного фрагменту” встановлюється рівним “0”, коли фрагмент є останнім. Нульовий індекс не є дозволеним в таблиці управління фрагментацією.

Важливо зазначити, що розмір фрагменту в блоках, на відміну від розміру файлу в блоках в таблиці адресації, включає в себе кінцевий блок, якщо фрагмент його містить. Врахування кінцевого блока є задачею таблиці адресації та інструментів, що працюють з нею; задачею таблиці фрагментації є лише розміщення файлу в декількох позиціях накопичувача одночасно.

Фактичного обмеження на кількість можливих фрагментів немає, але в реалізацію включено оптимізаційні процедури, задачею яких є якомога швидше звільнення таблиці фрагментації, та які не повинні допустити фрагментацію файлу на більш ніж три частини, виконавши натомість переміщення існуючих файлів щоб зменшити фрагментацію даного.

Вважається, що за нормальних умов таблиця управління фрагментації є пустою. За замовчуванням для її записів резервується лише один блок накопичувача.

3.10. Таблиця обліку записів

Таблиця обліку записів необхідна для того, щоб слідкувати за кількістю записів в кожен блок накопичувача, побудованого з використанням flash-пам’яті. Ця таблиця має дуже простий вигляд та індексує кожен блок накопичувача, маючи в якості значення число записів даного блоку. Для оптимальної роботи таблиці обліку записів необхідно встановлювати розмір внутрішнього блоку тег-орієнтованого файлового сховища рівним розміру фізичного блоку flash-пам’яті, за яким виконується перезапис.

Формат запису для таблиці обліку записів наведено в таблиці 3.14.

Таблиця 3.14 — Формат запису для таблиці обліку записів

Відносна адреса	Розмір	Призначення
`0x00`	32	Число записів до відповідного блоку

Формат таблиці обліку записів є дуже простим, адже пошкодження даних, збережених в цій таблиці, ніколи не є критичним. Взагалі, процедури обліку записів мають евристичний характер (насамперед необхідно розуміти, що при побудові сховища вважається, що у всі блоки ще не виконувалось записів, що, звісно, не гарантується), і можуть бути вимкнуті при побудові сховища для вивільнення додаткового простору накопичувача.

4. АНАЛІЗ РЕЗУЛЬТАТІВ

4.1. Дослідження швидкодії вибірки файлів

Дослідження результатів роботи здійснено за всіма напрямками оптимізації. Найпершим та очевидним напрямком є вибірка файлів за запитами різної складності.

Порівняння виконується з реалізаціями інших файлових систем та сховищ, що працюють на рівні користувача (за допомогою FUSE чи інших технологій), написаних мовою програмування Python 3, щоб уникнути спотворення результатів через продуктивність певної мови програмування чи бібліотек.

Дослід з кожним набором параметрів виконується 10 разів, як результат приймається середнє арифметичне значення усіх результатів. Результати, що відхиляються від середнього значення на більше, ніж 10% (за умови, що таке відхилення не спостерігається більше одного разу), відкидаються як спотворені через дію не врахованих факторів (як то: не враховане навантаження досліджувального стенду, виконання операційною системою синхронізації тощо).

В рамках досліду змінюються наступні параметри:

— кількість файлів, збережених у сховищі;

— кількість файлів, що вибираються;

— середній розмір файлів, що вибираються;

— складність запиту на вибірку;

— спосіб, яким виконується вибірка, для ієрархічної файлової системи чи сховища.

Очікується, що:

— залежність між кількістю файлів, збережених у файловому сховищі (або ж його загальним об’ємом) та швидкодією сховища спостерігатися не буде;

— залежність між кількістю файлів, що вибираються, та швидкодією сховища буде спостерігатись: зі збільшенням кількості файлів, що вибираються, швидкодія буде зменшуватись;
— залежність між середнім розміром файлу, що вибирається, та швидкодією сховища буде спостерігатись: зі збільшенням середнього розміру файлу, швидкодія буде зменшуватись;
— залежність між складністю запиту на вибірку та швидкодією сховища буде спостерігатись: зі збільшенням складності запиту, швидкодія буде зменшуватись;
— при збільшенні кількості файлів, що вибираються, швидкодія ієрархічного файлового сховища чи системи буде зменшуватись швидше, ніж тег-орієнтованого;
— при збільшенні складності запиту на вибірку швидкодія ієрархічного файлового сховища чи системи буде зменшуватись швидше, ніж тег-орієнтованого;
— незалежно від обраного способу виконання вибірки, тег-орієнтоване файлове сховище буде показувати менше, ніж ієрархічне, падіння швидкодії при ускладненні умов.

Так як структура всіх розповсюджених не спеціалізованих (тобто загального призначення) файлових сховищ та систем (як ієрархічних, так і інших за принципом роботи), в тому числі структура тег-орієнтованого файлового сховища, побудованого за запропонованим в даній роботі методом, дозволяє вибірку конкретного файлу, в загальному випадку не зважаючи на фізичну позицію його даних на накопичувачі, не очікується спостереження залежності між кількістю файлів (чи об’ємом зайнятого простору накопичувача) та швидкодією сховища чи системи, адже єдиний ефект, до якого призводить зміна цього параметру роботи — зміна фізичного положення даних конкретного файлу на накопичувачі.

Збільшення кількості файлів, що вибираються, натомість, неминуче призводить до виконання більшої кількості операцій пошуку конкретного файлу на накопичувачі, що займають не нульовий час. Отже, збільшення кількості таких операцій повинно збільшити час виконання вибірки загалом, тобто зменшити швидкодію файлового сховища чи системи.

Збільшення складності запиту на вибірку потребує більш глибокого пошуку, вирізнення ознак/тегів файлу та виконання додаткових множинних дій над ними, в тим більшому об’ємі, чим більша складність (кількість термів) запиту. Отже, обробка складнішого запиту має виконуватись довше, і відповідно при використанні складніших запитів швидкодія сховища буде зменшуватись.

Спосіб виконання вибірки в рамках ієрархічної файлової системи визначає, як саме виконується вибірка файлів за запитом. В рамках дослідження розглядається три способи виконання вибірки: вручну (ніяких додаткових інструментів чи алгоритмів не застосовується, оператор виконує пошук потрібних файлів самостійно, використовуючи лише стандартні засоби ієрархічної файлової системи чи сховища), з використанням пошукових інструментів (таких як GNU find) чи з використанням тег-орієнтованих інструментів. Очевидно, у випадку вибірки вручну підвищення швидкодії тег-орієнтованого файлового сховища, побудованого на накопичувачі, у порівнянні буде найбільш помітним, адже виконується фактично пряма комп’ютеризація даного процесу. У випадку вибірки з використанням пошукових інструментів швидкодія тег-орієнтованого файлового сховища, побудованого на накопичувачі, очікується вищою, адже не потрібно виконувати повний перебір всіх доступних файлів, як це роблять пошукові утиліти типу GNU find. У випадку вибірки з використанням тег-орієнтованих інструментів очікується незначна різниця у швидкодії між ієрархічним файловим сховищем або системою та файловим сховищем, побудованим на

накопичувачі, зумовлена тіснішою інтеграцією тег-орієнтованих інструментів пошуку з останнім (фактично, у такому випадку ці інструменти є частиною самого сховища і стандартними для нього засобами).

В рамках першого етапу дослідження порівняно затрати часу людини (оператора) на роботу зі сховищем з використанням різних можливих способів виконання вибірки (для ієрархічних файлових сховищ та систем — ручний, з використанням пошукових інструментів та з використанням тег-орієнтованих інструментів; для тег-орієнтованого файлового сховища — лише з використанням тег-орієнтованих інструментів).

Результати дослідження наведено в таблиці 4.1. Складність запиту вимірюється у кількості умов (тегів). Час взаємодії при вибірці вручну подано як “РВ”. Час взаємодії при використанні пошукових інструментів подано як “ПІ”. Час взаємодії при використанні тег-орієнтованих інструментів подано як “ТОІ”. Час взаємодії при використанні тег-орієнтованого файлового сховища подано як “ТОС”.

У випадках, коли час взаємодії залежить від кількості файлів у сховищі, які необхідно перевірити (власне, у випадку вибірки вручну), вказується не конкретний час взаємодії, а спостережена залежність між часом взаємодії та кількістю файлів, що зберігаються у файловому сховищі або системі.

Таблиця 4.1 — Результати дослідження часу взаємодії

Складність запиту	РВ, с	ПІ, с	ТОІ, с	ТОС, с
1	0,13 × N	3,29	2,79	2,76
2	0,76 × N	7,41	4,62	4,64
5	2,45 × N	21,98	10,18	10,06

З дослідження очевидно, що використання вибірки вручну не є доцільним ніколи, окрім випадків, коли потрібно переглянути незначний об’єм файлів (“незначним” можна вважати об’єм файлів до 10) на відповідність невеликій кількості ознак (перевірка на не більше, ніж 2 ознаки, є доцільною). В усіх інших випадках на вибірку вручну оператором буде затрачено значно більше часу, ніж на будь-який з інших розглянутих способів вибірки., адже лише час вибірки вручну залежить від кількості файлів, що потрібно переглянути, а не тільки від складності запиту.

Серед автоматизованих методів вибірки можна спостерігати значну розбіжність у часі взаємодії з пошуковими та тег-орієнтованими інструментами. Натомість, тег-орієнтовані інструменти показують однакові в межах похибки значення часу взаємодії. Це викликано поганою пристосованістю пошукових інструментів до задачі, для вирішення якої вони в даному дослідженні використовуються (будучи не спеціалізованими, пошукові інструменти не є оптимізованими для пошуку саме за тег-подібними ознаками).

Виходячи з результатів дослідження часу взаємодії, в подальшому ручний спосіб вибірки не досліджується як очевидно недоцільний.

Параметр кількості файлів, що вибираються, вирішено почати з 1000, адже за менших значень результати вимірів є неоднозначними і сильно спотворюються електронним шумом досліджувального стенду.

В таблицях результатів загальну кількість файлів у сховищі подано як “КФЗ”; кількість файлів, що вибираються запитом — “КФВ”; середній розмір файлу — “Розмір”; час, затрачений на вибірку з ієрархічного сховища з використанням пошукових інструментів (окрім часу взаємодії з оператором, що досліджувався окремо) — “ЧПІ”; час, затрачений на вибірку з ієрархічного сховища з використанням тег-орієнтованих

інструментів — “ЧТОІ”; час, затрачений на вибірку з тег-орієнтованого сховища (аналогічно) — “ЧТОС”.

Результати першого етапу дослідження наведено у таблиці 4.2. Використовуються запит на 1 ознаку.

Таблиця 4.2 — Результати дослідження швидкодії вибірки за запитом І

КФЗ	КВ	Розмір, байт	ЧПІ, с	ЧТОІ, с	ЧТОС, с
500000	1000	4096	1,478	0,267	0,069
500000	10000	4096	2,152	0,988	0,801
500000	50000	4096	4,838	3,772	3,533
500000	100000	4096	8,701	8,202	6,813
500000	200000	4096	16,029	13,842	13,813
2000000	1000	4096	5,68	0,871	0,074
2000000	10000	4096	6,38	1,567	0,676
2000000	50000	4096	9,263	4,569	3,583
2000000	100000	4096	12,433	8,682	7,46
2000000	200000	4096	19,894	15,262	14,808
500000	1000	409600	8,693	7,503	7,3
500000	10000	409600	74,34	73,2	72,931
500000	50000	409600	366,695	364,897	364,774
500000	100000	409600	730,932	729,606	730,613
500000	200000	409600	1460,824	1459,498	1459,2
2000000	1000	409600	12,899	8,101	7,3
2000000	10000	409600	78,535	73,825	73,047
2000000	50000	409600	370,729	365,631	365,27
2000000	100000	409600	736,223	730,873	730,642
2000000	200000	409600	1464,943	1459,761	1459,335

Результати другого етапу дослідження наведено у таблиці 4.3. Використовуються запит на 20 ознак.

Таблиця 4.3 — Результати дослідження швидкодії вибірки за запитом II

КФЗ	КВ	Розмір, байт	ЧПІ, с	ЧТОІ, с	ЧТОС, с
500000	1000	4096	1,494	0,303	0,082
500000	10000	4096	2,376	1,128	0,853
500000	50000	4096	7,036	5,688	4,237
500000	100000	4096	11,693	11,438	8,327
500000	200000	4096	24,043	19,353	18,055
2000000	1000	4096	5,7	0,894	0,084
2000000	10000	4096	6,599	1,875	0,812
2000000	50000	4096	10,555	5,685	3,879
2000000	100000	4096	15,946	11,058	7,699
2000000	200000	4096	25,723	23,282	15,595
500000	1000	409600	11,71	10,495	8,298
500000	10000	409600	104,331	103,114	82,921
500000	50000	409600	515,998	515,621	414,838
500000	100000	409600	1032,324	1029,828	830,531
500000	200000	409600	2062,954	2060,045	1660,512
2000000	1000	409600	15,898	11,092	8,297
2000000	10000	409600	108,498	103,781	82,924
2000000	50000	409600	521,047	515,975	415,237
2000000	100000	409600	1036,583	1030,279	830,042
2000000	200000	409600	2067,643	2059,433	1659,9

За даними з таблиць, для більшої наглядності результатів дослідження, було побудовано графіки залежностей швидкодії ієрархічного та тег-орієнтованого сховищ.

Графіки залежності часу виконання вибірки від загальної кількості збережених до сховища файлів наведено на рис. 4.1.

Можна спостерігати, що графіки паралельні для часових характеристик у всіх трьох випадках. Тобто, залежність відсутня. Це цілком збігається з очікуваним результатом: залежність між даними величинами

не повинна спостерігатись, адже будь-яке файлове сховище чи система оптимізується для вибірки конкретного файлу з будь-якого положення на накопичувачі за константний час, наскільки це можливо. А кількість файлів, збережена у сховищі, може впливати лише на положення конкретного файлу.

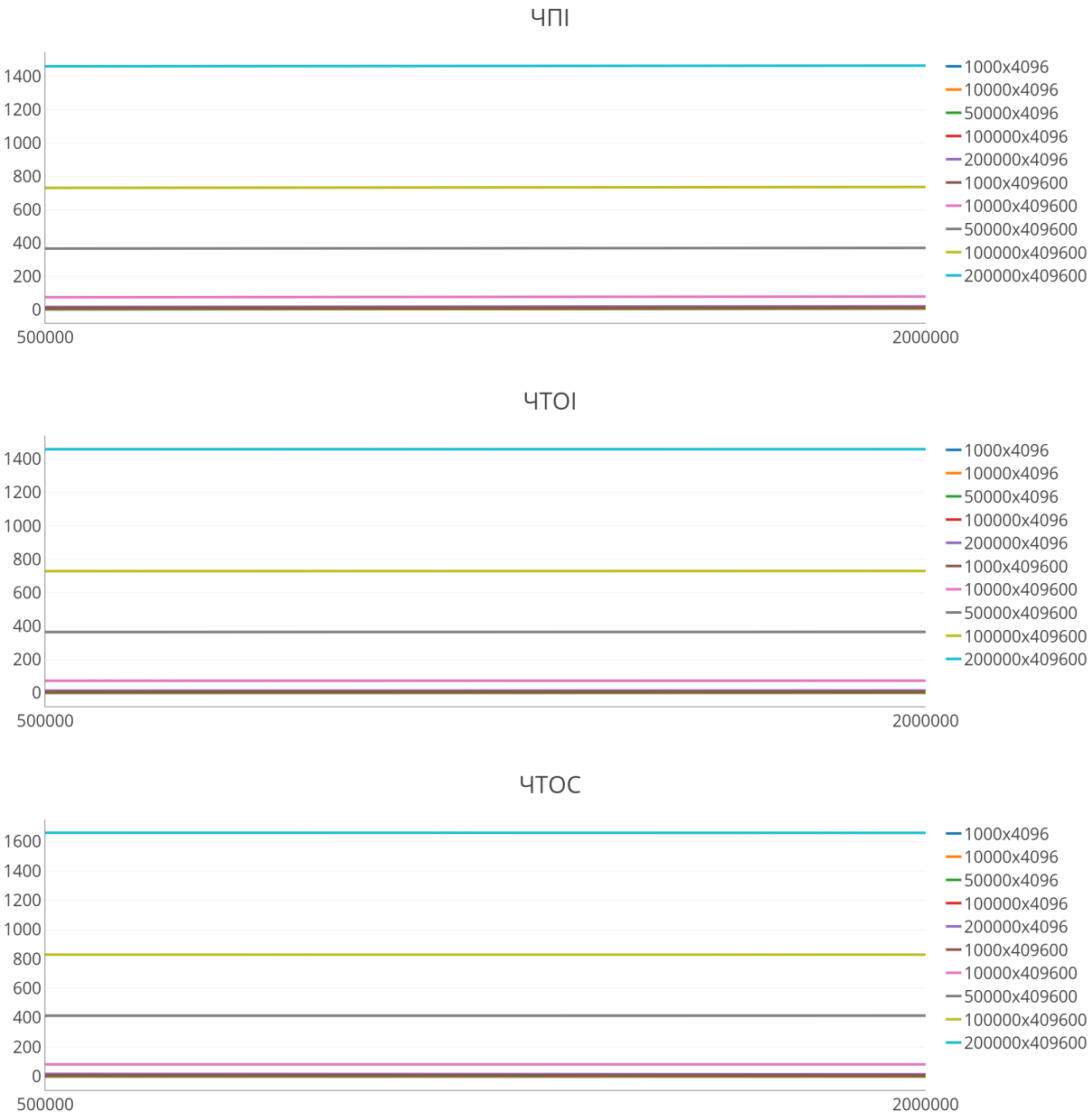

Рисунок 4.1 — Залежність часу виконання вибірки від загальної кількості файлів у сховищі

Незважаючи на те, що залежність між цими величинами не спостерігається, спостерігається зміна затраченого на вибірку часу на константну величину — близько 4.2 секунд у випадку використання інструментів пошуку та близько 0.6 секунд у випадку використання тег-орієнтованих інструментів з ієрархічним сховищем. Це пояснюється тим, що для виконання запиту необхідно (у випадку використання інструментів пошуку) здійснити повний обхід накопичувача чи (у випадку використання тег-орієнтованих інструментів) завантажити та проіндексувати базу даних відповідного тег-орієнтованого інструмента. Як довжина цього обходу, так і об’єм бази даних, що завантажується та індексується, залежать від загальної кількості файлів у сховищі. Натомість, у випадку використання тег-орієнтованого файлового сховища така зміна не спостерігається, адже не потрібно ні виконання повного обходу, ні завантаження бази даних (всі необхідні дані вже наявні у пам’яті з моменту індексації накопичувача).

Графіки залежності часу виконання вибірки від середнього розміру файлів, що вибираються, наведено на рис. 4.2.

На відміну від випадку з загальною кількістю файлів у сховищі, можна спостерігати чітку залежність: час вибірки збільшується зі збільшенням розміру файлу (через відсутність залежності від загальної кількості файлів у сховищі, графіки з різними загальними кількостями, але однаковою кількістю, що вибирається, накладаються попарно і не можуть бути побачені обидва), притому збільшується тим стрімкіше, чим більше файлів вибирається. Це зумовлено тим, що для вибірки необхідно прочитати файли з накопичувача (обмеження швидкодією самого накопичувача — максимальна швидкість читання є чітко визначеною та обмеженою), і чим більше таких файлів, тим довше триває процес читання. В обраному масштабі здається, що всі графіки починаються з однієї точки, адже збільшення середнього розміру файлу в 100 разів фактично збільшило кожний час вибірки в таке ж число разів.

Так як залежність часу вибірки від середнього розміру файлу зумовлена лише швидкодією самого накопичувача, в рамках цієї залежності не можна виділити переваги чи недоліки тег-орієнтованого файлового сховища в порівнянні з ієрархічним.

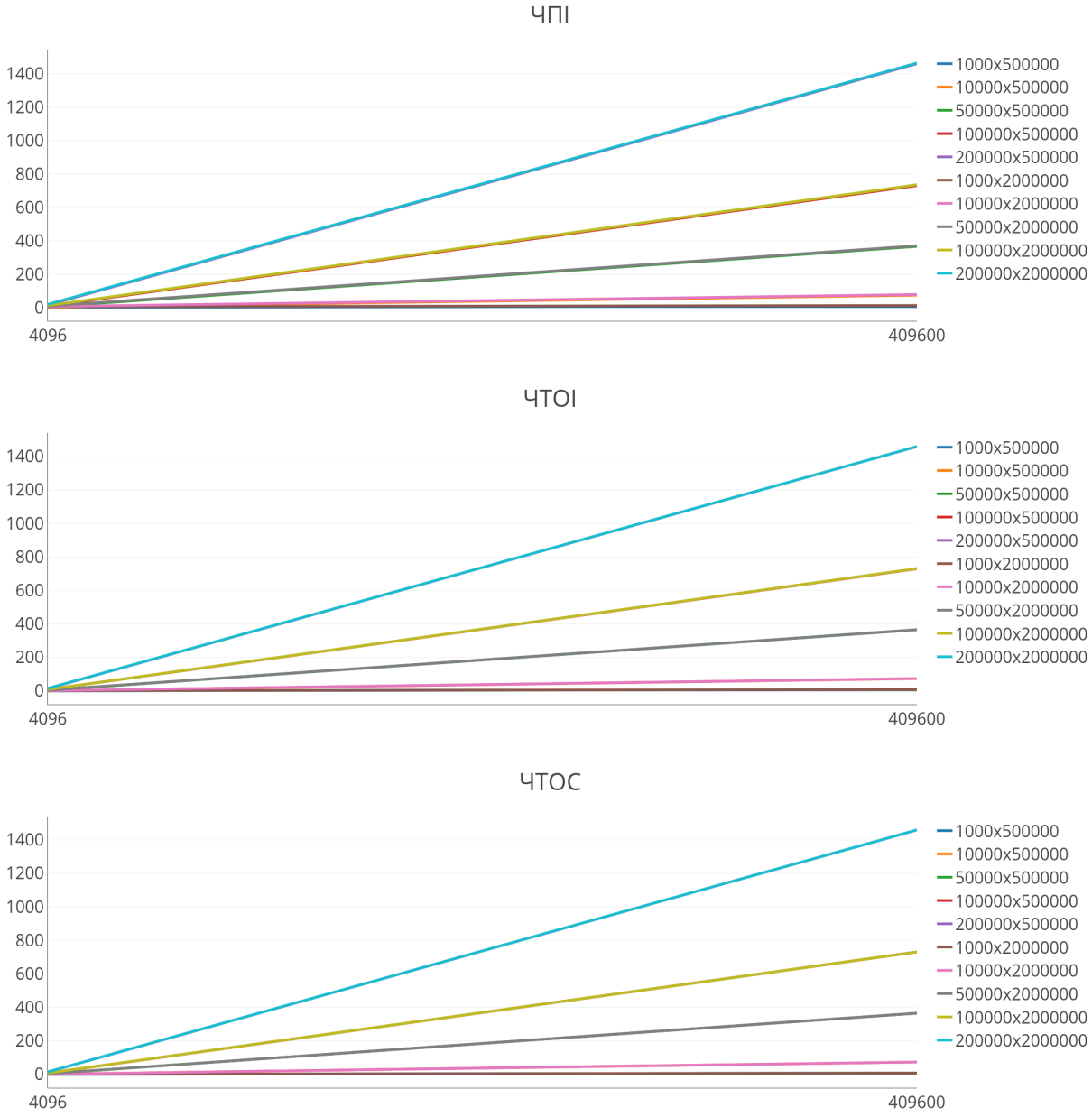

Рисунок 4.2 — Залежність часу виконання вибірки від середнього розміру файлів, що вибираються

Графіки залежності часу виконання вибірки від кількості файлів, що вибираються, наведено на рис. 4.3. Зважаючи на нерелевантність загальної

кількості файлів та їх середнього розміру (як можна побачити з аналізу відповідних графіків) до залежності часу виконання вибірки до кількості файлів, що вибираються, на цих графіках використовуються дані з наборів, де зазначена загальна кількість файлів 2000000 та розмір файлу 4096 байт.

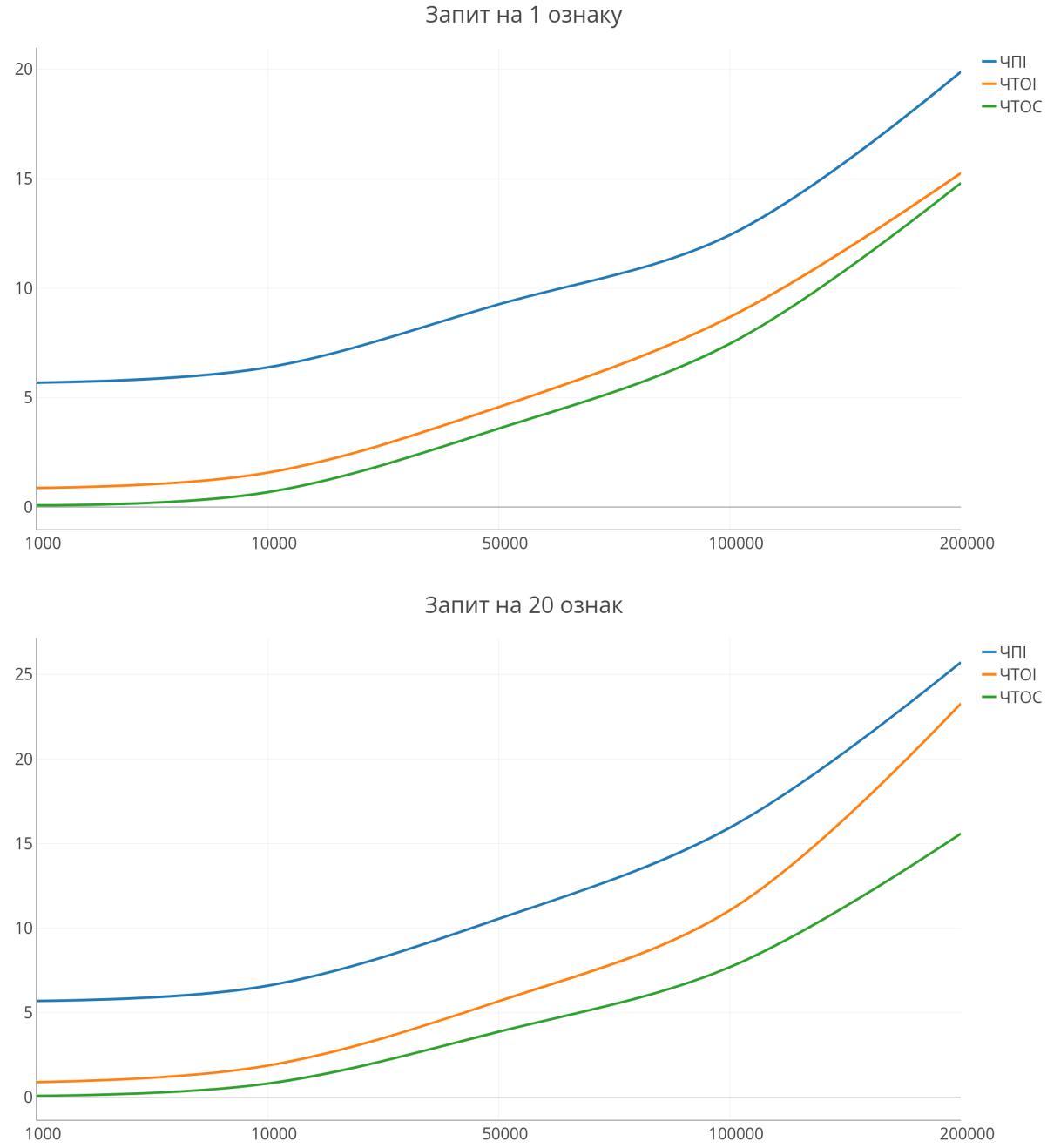

Рисунок 4.3 — Залежність часу виконання вибірки від кількості файлів, що вибираються

Можна спостерігати чітку залежність часу вибірки від кількості файлів, що вибираються. Крім того, добре помітно, як при використанні тег-орієнтованого файлового сховища завжди на вибірку потрібно менше часу, ніж при використанні ієрархічного, незалежно від способу вибірки, який застосовується.

В умовах простих запитів (одна ознака), проте, використання тег-орієнтованих інструментів для вибірки з ієрархічного сховища дозволяє досягнути рівня швидкодії, майже ідентичного до поміченого при використанні тег-орієнтованого файлового сховища. Але для складніших запитів це перестає бути справедливим: з ускладненням запиту для ієрархічного файлового сховища завжди потрібно більше часу на обробку кожного файлу, ніж для тег-орієнтованого.

Це зумовлено тим, що, на відміну від тег-орієнтованого сховища, що напряму побудоване на асоціативних зв’язках та таблицях асоціативної індексації інформації, інструменти ієрархічного сховища вимушені будувати ці зв’язки та виконувати індексацію у процесі роботи, використовуючи для цього субоптимальні структури даних та алгоритми порівняння.

Наприклад, розглянуті в рамках дослідження тег-орієнтовані інструменти використовують збережену у файл власну базу даних, що вимагає побудови та оновлення незалежно від операцій з файлами, тоді як для тег-орієнтованого файлового сховища будь-яка операція з файлами вже фактично є оновленням такої внутрішньої “бази даних” та не може відбутися без вищезгаданого оновлення. Розглянуті ж інструменти пошуку взагалі не є пристосованими до саме задачі асоціативної вибірки інформації, а є інструментом загального призначення, які, як відомо, не в загальному випадку не здатні показувати такі ж результати, як спеціалізовані інструменти, або кращі.

Також очевидно, що затрачений на вибірку час при використанні обох типів сховищ (і ієрархічного, і тег-орієнтованого) завжди відрізняється на майже незмінну величину незалежно від кількості файлів, що вибираються (при збільшенні складності запиту ця різниця поступово збільшується при ускладненні запиту, але не дуже виразно). Це зумовлено тим, що при використанні ієрархічного сховища для обробки запиту необхідно виконати повний обхід та індексацію всіх файлів, збережених на накопичувачі. Це залишається справедливим для кожного запиту. Час виконання цієї операції не залежить від кількості файлів, що вибираються, а лише від загальної кількості збережених у сховищі файлів.

Загалом результати дослідження швидкодії вибірки показують, що є особливо доцільним використання тег-орієнтованого файлового сховища у двох випадках:

— виконуються, в середньому, вибірки, що охоплюють невелику частину збережених у сховищі файлів;
— вибірки виконуються з використанням складних запитів (більше, ніж одна умова).

Перший випадок випливає з факту наявності незмінної різниці на час початкової обробки у ієрархічному сховищі між показниками ієрархічного та тег-орієнтованого сховищ. Використання тег-орієнтованого файлового сховища є доцільним навіть за умови вибірки за простими запитами, якщо вибирається об’єм файлів, у рамках якого ця незмінна різниця ще є значущою.

Якщо вважати значущою частину незмінної різниці в часі вибірки, рівну 10% від загального часу вибірки, можна записати наступну умову доцільності використання тег-орієнтованого файлового сховища:

$$\frac{S \cdot A_S}{Bw} \leq 10\, \Delta t_i \cdot A_T, \qquad (4.1)$$

де:

S — середній розмір файлу;

A_S — кількість файлів, що вибираються;

Bw — швидкість читання файлу зі сховища;

Δt_i — час індексації одного файлу;

A_T — загальна кількість файлів у сховищі.

З експериментальних даних можна отримати значення швидкості читання з накопичувача 53 МіБ/с та затримки на індексацію кожного файлу (із загальної кількості, що зберігаються у сховищі) порядку $2{,}8 \times 10^{-6}$ с. Якщо підставити ці значення у (4.1) та виконати скорочення, отримуємо умову для даних накопичувача та ієрархічного сховища:

$$A_S \leq 1556 \frac{A_T}{S} \qquad (4.2)$$

Підставляючи в (4.2) експериментальні дані (наприклад, S = 4096, A_T = 500000), отримуємо значення A_S, при якому є доцільним використання тег-орієнтованого файлового сховища, 189941 файл або менше, що підтверджується дослідом (при досягненні кількості файлів, що вибираються, більшої за це значення — 200000 — спостерігається однакова продуктивність обох типів сховищ).

Підставляючи інакший набір експериментальних даних (S = 409600, A_T = 2000000), отримуємо значення A_S, при якому є доцільним використання тег-орієнтованого файлового сховища, 7597 файлів або менше, що також збігається зі спостереженими даними.

Другий випадок випливає з факту наявності додаткової затримки на обробку кожної додаткової ознаки запиту при перевірці кожного файлу. Так як механізм перевірки ознак (тегів) є інтегрованим до тег-орієнтованого

сховища та, загалом, природнім шляхом взаємодії з таким сховищем, для швидкодії якого сховище оптимізувалося, ця затримка є меншою у випадку використання тег-орієнтованого файлового сховища. Тому при використанні запитів на велику кількість ознак умова (4.1) вже не може бути застосована. Використовувати тег-орієнтовані файлові сховища рекомендується завжди, коли очікується часта вибірка за складними запитами, адже такі сховища завжди покажуть кращий, ніж ієрархічні, результат при вибірці за складним запитом.

4.2. Дослідження використання простору накопичувача

Другим напрямком оптимізації, що досліджується, є об’єм вільного простору накопичувача, що файлова система чи сховище робить доступним для збереження користувацьких файлів.

Порівняння виконується з розповсюдженою файловою системою ext4, що була побудована на накопичувачі з використанням стандартних налаштувань. Для порівняння тег-орієнтоване файлове сховище, побудоване на накопичувачі за запропонованим методом, було побудоване як з підтримкою пакування кінцевих частин файлів, так і без такої.

В рамках досліду змінюються наступні параметри:

— кількість збережених у сховищі файлів;

— середній розмір збереженого файлу.

Очікується, що:

— при відсутності будь-яких файлів тег-орієнтоване сховище надасть для користувача більше вільного простору, ніж файлова система ext4;

— кожен доданий файл буде займати відповідну його розміру, округленому до розміру блоку вгору, частину вільного простору у випадку файлової системи ext4;

— кожен доданий файл буде займати відповідну його розміру, округленому до розміру блоку вгору, частину вільного простору у випадку тег-орієнтованого файлового сховища без використання пакування кінцевих частин файлів;

— кожен доданий файл буде займати меншу, ніж його розмір, округлений до розміру блоку вгору, частину вільного простору у випадку тег-орієнтованого файлового сховища з використанням пакування кінцевих частин файлів;

— при використанні тег-орієнтованого сховища можливе раптове зменшення доступного простору на величину, не пов’язану з розміром файлу.

Зменшення зайнятого кожним файлом простору накопичувача є основним призначенням пакування кінцевих частин файлів. Цей ефект досягається завдяки збереженню кінцевих частин двох файлів, що є меншими за розмір блоку накопичувача, до одного блоку. У найкращому випадку (кожен файл має кінцеву частину, рівну за розміром половині розміру блоку) можна очікувати зменшення зайнятого простору накопичувача завдяки використанню пакування кінцевих частин файлів на половину розміру блоку для кожного збережного файлу, тобто на $F \cdot \frac{Bs}{2}$ байт, де F — кількість збережених файлів, Bs — розмір блоку у байтах.

У випадку, коли пакування кінцевих частин файлу не використовується, для збереження кінцевої частини файлу завжди використовується повний блок, незалежно від її розміру.

Коли у сховищах не зберігається жодного файлу, тег-орієнтоване файлове сховище надає користувачу більше вільного простору завдяки меншому об’єму метаданих, що зберігаються у такому випадку. Файлова система ext4 резервує місце для метаданих у момент побудови файлової системи на накопичувачі, тоді як тег-орієнтоване файлове сховище не

виконує резервування наперед, натомість дозволяючи зміну розміру управляючих таблиць та структур даних для збереження мета-даних, коли це потрібно.

Саме через можливість зміни розміру управляючих структур даних можливе раптове зменшення доступного простору: це може відбутись внаслідок перебалансування управляючих структур даних, щоб вміщувати інформацію про більшу кількість файлів та тегів, тощо.

Результати дослідження подано у таблиці 4.4. Загальну кількість файлів у сховищі подано як “КФЗ”; середній розмір файлу — “Розмір”; об’єм вільного простору накопичувача для файлової системи ext4 — “Oext4”, для тег-орієнтованого файлового сховища без використання пакування кінцевих частин файлу — “OTOC”, для тег-орієнтованого файлового сховища з використанням пакування кінцевих частин файлу — “OTOCпак”. Колонки “% OTOC” та “% OTOCпак” показують, наскільки більше простору накопичувача надано користувачу у відсотках в порівнянні з наданим файловою системою ext4.

Розмір файлу вказано в байтах. Об’єм вільного простору накопичувача вказано в блоках. Блок є рівним 4096 байт.

Таблиця 4.4 — Результати дослідження використання простору

КФЗ	Розмір	Oext4	OTOC	% OTOC	OTOCпак	% OTOCпак
0	0	49218750	49897600	1,38	49897600	1,38
1000	2048	49217750	49896600	1,38	49897100	1,38
1000	4096	49217750	49896600	1,38	49896600	1,38
1000	6144	49216750	49895600	1,38	49896100	1,38
1000	200000	49169750	49848600	1,38	49848772	1,38
100000	2048	49118750	49797600	1,38	49847600	1,48
100000	4096	49118750	49797600	1,38	49797600	1,38
100000	6144	49018750	49697600	1,38	49747600	1,49

100000	200000	44318750	44997600	1,53	45014788	1,57
1000000	2048	48218750	48897600	1,41	49397600	2,44
1000000	4096	48218750	48897600	1,41	48897600	1,41
1000000	6144	47218750	47897600	1,44	48397600	2,50
1000000	200000	218750	897600	310,33	1069475	388,90

Результати дослідження відповідають очікуваним. При незначній кількості файлів у сховищі спостерігається лише приріст вільного простору на константну величину при використанні тег-орієнтованого сховища завдяки меншому об’єму метаданих, що зберігаються (ext4 резервує 256 байт на кожний можливий файл при побудові, тобто при побудові файлової системи ext4 задається максимальна можлива кількість файлів у ній, тоді як тег-орієнтоване файлове сховище не резервує простір накопичувача для збереження метаданих, поки в ньому немає потреби, та до того ж резервує близько 128 байт на файл у середньому). При значній кількості файлів у сховищі цей константний приріст вільного простору стає, у відносних величинах, особливо помітним.

Також на величинах файлів, не кратних розміру блока (4096 байт), можна помітити ефект від пакування кінцевих частин як майже подвоєння відносної величини вільного простору у окремих випадках.

Результати дослідження використання дискового простору ієрархічною файловою системою та тег-орієнтованим файловим сховищем показують, що є особливо доцільним використовувати тег-орієнтоване файлове сховище за умов:

— роботи з великою кількістю файлів;

— середнього розміру файлів порядку десятків блоків і менше;

— не кратних розміру блока розмірах файлів.

Також є доцільним використання тег-орієнтованого файлового сховища у випадку роботи з невеликою кількістю файлів значного розміру, адже для збереження їх метаданих буде використано менше простору (у порівнянні з, наприклад, ієрархічною файловою системою ext4).

4.3. Дослідження використання ресурсу записів накопичувача

Третім напрямом оптимізації, що досліджується, є використання ресурсу записів накопичувача, побудованого з використанням flash-пам’яті.

Для дослідження використовується критерій найбільшої кількості перезаписів блока. Щоб його визначити, необхідно виконати запит до контролера накопичувача щодо кількості перезаписів кожного з фізичних блоків накопичувача, та вибрати з цього масиву даних найбільше значення.

Порівняння виконується з тег-орієнтованим сховищем tagme-file, побудованим з використанням проміжної ієрархічної файлової системи ext4. Тег-орієнтоване файлове сховище побудовано на накопичувачі з використанням всіх відповідних опцій (вимкнено журнал тощо).

Обидва сховища використовуються у штучно створених умовах: одним днем використання вважається запис 1000 файлів сумарним об’ємом 1 ГіБ, з яких 100 є новими (при цьому 100 старих файлів випадково видаляються) і 900 оновленими.

В рамках досліду змінюються наступні параметри:

— доля завантаження сховища (відсоткове значення, що відображає відношення зайнятого простору сховища до його загального об’єму);

— час використання.

Очікується, що:

— в процесі використання обох сховищ буде спостерігатись зростання значення найбільшої кількості перезаписів блока;

— зростання буде більш помітним та яскравим у випадку ієрархічної файлової системи;

— на певному етапі зростання припиниться у випадку тег-орієнтованого файлового сховища.

Так як в процесі експлуатації сховища неминуче виконуються записи, зростання найбільшої кількості перезаписів має відбуватися також неминуче.

Ієрархічна файлова система виконує перезаписи мета-даних в одному й тому ж фізичному місці накопичувача, тому якнайменше цей блок буде отримувати найбільше перезаписів. Через високу активність одного блока накопичувача буде швидко збільшуватись і значення найбільшої кількості перезаписів.

У певний момент кількість перезаписів ключових блоків тег-орієнтованого файлового сховища досягне критичного значення, і відповідні структури даних будуть переміщені у нове місце накопичувача. Це призведе до спостереженого припинення зростанння значення найбільшої кількості перезаписів, адже у блок, що надає це значення, більше не будуть виконуватись записи завдяки перенесенню відповідних структур даних. В подальшому буде спостерігатись значно повільніше, ніж у випадку ієрархічної файлової системи, зростання найбільшої кількості перезаписів внаслідок розповсюдження записів по всьому накопичувачу, що виконується тег-орієнтованим файловим сховищем.

Результати дослідження подано у таблиці 4.5. Завантаження сховища подано як “Завантаження”, у відсотках. Час використання як “Час”, у днях. Найбільшу кількість перезаписів для ієрархічного сховища як ПЗІ. Найбільшу кількість перезаписів для тег-орієнтованого сховища як ПЗТО.

Результати дослідження загалом відповідають очікуваним. Не відповідає очікуванням лише спостереження, що накопичення перезаписів для тег-орієнтованого сховища до досягнення критичного значення показувало загалом той самий рівень зростання, як і накопичення перезаписів для ієрархічного сховища. Це викликано тим, що в еталонній

Таблиця 4.5 — Результати дослідження кількості перезаписів

Завантаження	Час	ПЗІ	ПЗТО
0	0	0	0
5	10	736	649
50	10	794	653
95	10	1217	1301
5	200	14673	12964
50	200	15786	13078
95	200	24823	25916
5	1000	72503	50000
50	1000	78019	50000
95	1000	123215	50000

реалізації тег-орієнтованого файлового сховища, яка досліджується, не реалізовані всі необхідні модифікації щодо зменшення кількості метаданих, що записуються. Крім того, було вимкнуто журнал, а кешування в пам’яті системи, очевидно, не дає достатніх результатів.

Можна чітко спостерігати досягнення критичного значення перезаписів для блоку мета-даних у тег-орієнтованого файлового сховища. Це значення встановлене в 50000 перезаписів; після його досягнення перезаписів блоку, що до тих пір найбільш активно перезаписувався, більше не виконувалось, а дані були перенесені у нове місце.

Також можна спостерігати значно вищу швидкість зростання найбільшої кількості перезаписів у випадках, коли сховище завантажене майже повністю. Це зумовлено необхідністю частого перезапису та переміщення старих файлів, щоб проводити операції з новими — добре відомий та вивчений феномен для пристроїв, побудованих на базі flash-пам’яті.

Результати дослідження найбільшої кількості перезаписів блока при використанні накопичувача з ієрархічною файловою системою та з тег-

орієнтованим файловим сховищем показують, що є особливо доцільним використовувати тег-орієнтоване файлове сховище з накопичувачами, в основі яких лежить flash-пам’ять, за умов:

— ввімкнення всіх відповідних оптимізацій;

— використанні накопичувача не повністю (необхідно залишати вільними хоча б 10% простору накопичувача);

— використанні накопичувача впродовж довгого часу.

Загалом, рекомендується використовувати тег-орієнтоване тегове сховище в умовах, коли запис та видалення даних виконується порівняно рідко, а вибірка — часто. Тег-орієнтоване файлове сховище показує себе найкраще саме при виконанні вибірки файлів, і майже не має переваг при записі чи видаленні. Внесення файлів до тег-орієнованого файлового сховища також потребує більше часу та зусиль від оператора, адже на відміну від ієрархічного файлового сховища чи системи, необхідно призначити відповідні теги до кожного файлу. Так як тег-орієнтоване файлове сховище найкраще показує себе при використанні значної кількості тегів та складних запитів, цей процес може займати значний період часу.

ВИСНОВКИ

В магістерській дисертації розглянуто проблематику використання файлових сховищ та систем, насамперед тег-орієнтованих, та проведений аналіз доцільності можливих методів та способів їх вирішення, розглянуто їх переваги та недоліки.

Створено та запропоновано метод побудови тег-орієнтованого сховища з використанням прямого доступу до накопичувача, що суміщує в собі кращі (за результатами аналізу) з розглянутих підходів до вирішення задач, що постають в процесі використання таких сховищ.

В роботі наводиться опис створеного методу, пояснення та обґрунтування прийнятих рішень щодо його складових та конкретних особливостей реалізації. Наводиться також опис запропонованої еталонної реалізації методу, структур даних та алгоритмів, що реалізують описаний у дисертації метод побудови файлового сховища в рамках даної еталонної реалізації.

Виконані дослідження доцільності використання запропонованого методу побудови тег-орієнтованого файлового сховища. В рамках досліджень використовувалась описана еталонна реалізація методу. Виконано аналіз результатів досліджень.

За результатами досліджень можна встановити, що використання тег-орієнтованого файлового сховища, побудованого за запропонованим методом, дозволяє в окремих випадках (а саме, велика кількість файлів збережена у сховищі, порівняно незначна кількість вибирається, використовується складний запит на декілька ознак) прискорити вибірку файлів в 20 і більше разів, тоді як у інших випадках досягається менше прискорення, але ніколи не спостерігається сповільнення вибірки в порівнянні з ієрархічними файловими сховищами та системами, а також

тег-орієнтованими сховищами, побудованими на базі ієрархічних файлових сховищ та систем.

Рекомендується використання тег-орієнтованого файлового сховища, побудованого за запропонованим методом, насамперед для управління архівами інформації або у інших подібних випадках, коли потрібно виконувати часті зчитування та вибірки, але порівняно рідки записи чи видалення. Ця рекомендація зумовлена тим, що, хоч швидкодія власне сховища не страждає під час запису чи видалення файлів, процес призначення великої кількості тегів до файлу вимагає значних затрат часу оператора.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Pate S. UNIX Filesystems / Steve Pate. — 1st edition. — Hoboken: John Wiley & Sons. — 2003. — 472 p.

2. Bar M. Linux File Systems / Moshe Bar. — 1st edition. — New York: McGraw-Hill Companies. — 2001. — 512 p.

3. Von Hagen, W. Linux Filesystems / William Von Hagen. — Carmel: Sams Publishing. — 2002. — 600 p.

4. Glampaolo, D. Practical File System Design / Dominic Glampaolo. — 1st edition. — Burlington: Morgan Kaufmann. — 1998. — 256 p.

5. Harbron, T. File Systems: Structures and Algorithms / Thomas Harbron. — 1st edition. — Upper Saddle River: Prentice Hall. — 1988. — 254 p.